1-2021

Model Extraction and Adversarial Attacks on Neural Networks Using Side-Channel Information

Tommy Li
txl2747@rit.edu

Model Extraction and Adversarial Attacks on Neural Networks Using Side-Channel Information

Tommy Li

Model Extraction and Adversarial Attacks on Neural Networks Using Side-Channel Information

Tommy Li

January 2021

A Thesis Submitted
in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in
Computer Engineering

R·I·T | Kate Gleason
College of ENGINEERING

Department of Computer Engineering

Model Extraction and Adversarial Attacks on Neural Networks Using Side-Channel Information

Tommy Li

Committee Approval:

Dr. Cory E. Merkel *Advisor* Date
Assistant Professor, Department of Computer Engineering

Dr. Raymond Ptucha Date
Associate Professor, Department of Computer Engineering

Dr. Marcin Łukowiak Date
Professor, Department of Computer Engineering

Acknowledgments

Dr. Cory E. Merkel

Dr. Raymond W. Ptucha

Dr. Dhireesha Kudithipudi

Mr. Richard A. Tolleson

Dr. Marcin Łukowiak

Ms. Lourdes Marquez-Douglas

Andrew R. Fountain

Brian Lab Members

Computer Engineering Department

To everyone who has gotten me here - no matter how big or small the help was.

Abstract

Artificial neural networks (ANNs) have gained significant popularity in the last decade for solving narrow AI problems in domains such as healthcare, transportation, and defense. As ANNs become more ubiquitous, it is imperative to understand their associated safety, security, and privacy vulnerabilities. Recently, it has been shown that ANNs are susceptible to a number of adversarial evasion attacks - inputs that cause the ANN to make high-confidence misclassifications despite being almost indistinguishable from the data used to train and test the network. This thesis explores to what degree finding these examples may be aided by using side-channel information, specifically power consumption, of hardware implementations of ANNs.

A blackbox threat scenario is assumed, where an attacker has access to the ANN hardware's input, outputs, and topology, but the trained model parameters are unknown. The extraction of the ANN parameters is performed by training a surrogate model using a dataset derived from querying the blackbox (oracle) model. The effect of the surrogate's training set size on the accuracy of the extracted parameters was examined. It was found that the distance between the surrogate and oracle parameters increased with larger training set sizes, while the angle between the two parameter vectors held approximately constant at 90 degrees. However, it was found that the transferability of attacks from the surrogate to the oracle improved linearly with increased training set size with lower attack strength.

Next, a novel method was developed to incorporate power consumption side-channel information from the oracle model into the surrogate training based on a Siamese neural network structure and a simplified power model. Comparison between surrogate models trained with and without power consumption data indicated that incorporation of the side channel information increases the fidelity of the model extraction by up to 30%. However, no improvement of transferability of adversarial examples was found, indicating behavior dissimilarity of the models despite them being closer in weight space.

Contents

List of Figures

List of Tables

Acronyms

AI

Artificial Intelligence

ANN

Artificial Neural Network

BNN

Binarized Neural Network

CNN

Convolutional Neural network

MLP

Multi-Layer Perceptron

ReLU

Rectified Linear Unit

SCA

Side-Channel Attack

Chapter 1

Introduction

1.1 Motivation

Artificial intelligence (AI) and subsequently ANNs have become increasingly popular in the past few years. Due to recent advancements in the developments of ANNs and better hardware, they are quicker to train and are able to perform a wide variety of tasks through the use of specialized neural network structures. Some pre-trained models are even available for public usage, skipping the potentially long training process. Applications that most people use on a daily basis, such as Google, Facebook, and Twitter, use neural networks to help curate content for the user. Common services that everyday people use, such as banking and medical applications, can utilize user information to formulate predictions, such fraud detection, overdraft predictions, and health problems. In both cases, neural networks are interfacing with personal and potentially sensitive information. Neural networks also face potential issues when it comes to safety and security, especially in applications such as automation. For example, self-driving cars may make an incorrect decision when presented with an adversarial example, which could lead to a serious accident.

If an attacker is able to access the inputs and outputs of a neural network, they can potentially reverse-engineer the functionality of the neural network and use it to generate adversarial examples, even without knowing the network architecture.

As shown in [5], Papernot et al. were able to create a near functionally equivalent neural network by observing the output labels to a given input. The substitute neural network could be to generate adversarial examples using algorithms such as the fast gradient sign method (FGSM) that was proposed by Goodfellow et al. [1].

If the adversary has more access to the network, they can go as far as reverse-engineer the various parameters of the network, like what Batina et al. [4] and Jagielski et al. [6] have done. Batina et al. were able to recover nearly all parameters with side-channel information - the information obtained through the implementation of a system. Jagielski et al. managed to exploit the faults of an activation function within the network to recover the weights of the network. If the attacker managed to produce an equivalent model in functionality and parameters, it would not be difficult to steal the information from things that may interface with the network, such as banking details, credit card numbers, medical history, and more.

The overall goal of this thesis is to analyze and incorporate side-channel information obtained from the network to produce a surrogate network that not only retains the functionality of the original neural network, but also has the highest accuracy in terms of the weights. The closer the weights of the surrogate network are, the easier it becomes for an adversary to transfer an attack from the surrogate network to cause a misclassification on the original network. Specifically, the objectives of this thesis are as follows:

1. Perform the state of the art model extraction and blackbox attack methods on hardware binarized neural networks.

2. Design a model that mimics the power for hardware binarized neural networks between subsequent inputs and develop a novel co-optimization method to match the functional behavior and power profile of a surrogate network to a blackbox oracle.

3. Compare the effectiveness of blackbox attacks derived from the surrogate model to state-of-the-art blackbox attack methods.

1.2 Document Structure

The remainder of this document is organized as follows: Chapter 2 gives a brief summary on neural networks, side-channel attacks, and summaries of related works. Chapters 3-5 contains the work done on the above objectives. Chapter 6 contains a summary of the future work that can be performed.

Chapter 2

Background and Related Work

2.1 Multi-Layer Perceptron and Binarization

A multi-layer perceptron (MLP) is a type of feed-forward neural network with three distinct types of layers - an input layer, one or more hidden layers, and an output later. The inputs and output sizes are based on the size of the input data and number of classes, respectively. The hidden layer may vary in the number of nodes in the layer, as well as the number of hidden layers. This can vary based on the complexity of the task at hand - generally more complex datasets warrant multiple hidden layers. Each node in the multi-layer perceptron, besides the input, uses a non-linear activation function (such as tanh, sigmoid, ReLU) and backpropagation for training. An example of a simple MLP is shown in Figure 2.1.

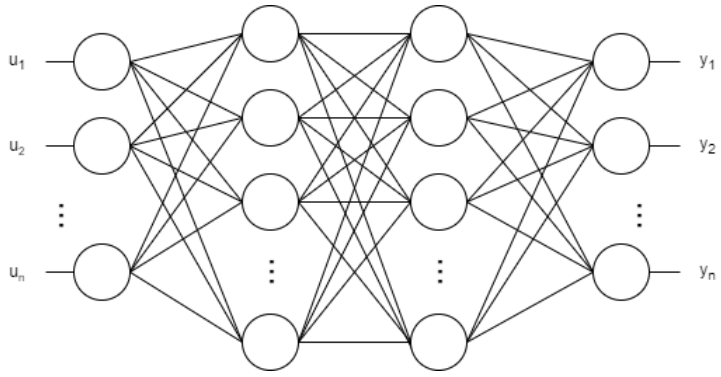


Figure 2.1: Simple Multi-Layer Perceptron

Binarization is the process of quantizing various parameters of a network, such as

the weights, biases, and activation functions. The most common binarization tactics are to set parameters to -1 or 1 and 0 and 1. Networks are binarized primarily to reduce the complexity and computational requirements of the implementation of the network. For the binary convolutional neural network (CNN) that was developed by Courbariaux et al. [7], the convolution operation, which is typically the most computationally intensive part of a CNN, can simply be done with an XNOR pop-count function. From a hardware point of view, this function takes fewer clock cycles overall to perform. However, the reduced complexity can lead to an overall loss of performance of the network, as information can be lost between layers due to the quantization. Courbariaux attempts to limit this by adding a batch normalization layer after each activation layer. With the proper architecture and optimizations, binarized neural networks are able to perform tasks such as image classification [7] [5] [8] and semantic segmentation [8]. Binary neural networks are also much easier to integrate with hardware, which is useful for obtaining side-channel information.

2.2 Attacks Against Neural Networks

2.2.1 Adversarial Attacks

Adversarial attacks are a set of attacks on neural networks with the intent of causing the target to deviate from its expected behavior during test time. Types of adversarial attacks range from data poisoning, evasion, and privacy attacks. Data poisoning is an attack where the adversary injects samples into the training set of the model, which can modify the behavior of the network during inference (or poisoning the data) [9] [10]. Evasion attacks are a series of attacks where the inputs to the network are modified in a manner to where a network misclassifies the sample during inference [11] [10]. As the networks targeted for model extraction have already been trained, evasion attacks will be the main focus, as data poisoning is not usually performed

outside of training.

Adversarial evasion attacks are performed on neural networks intentionally cause them to mispredict an input [1] [12]. This is done through the generation of adversarial examples, which are modified samples from a dataset that have small perturbations added to them. Several methods can be used to generate perturbations, but the most popular methods involve the use of gradients. Typically, these perturbations are small so that the modified input is relative close to the original. In the context of image classification, the perturbed image is generally indistinguishable from the original to the human eye. One of the most popular adversarial attacks is known as the fast gradient sign method (FGSM), which was introduced by Goodfellow et al. [1] in 2015. Their proposed method added a small perturbation to the input image using the gradient of the loss with respect to the input:

$$x_{adv} = x + \epsilon \times sgn(\Delta_x J(\theta, x, y)) \tag{2.1}$$

where x is the original input, y is the output label of x, J is the loss, ϵ is a factor to keep the perturbation minimal, and x_adv is the newly generated adversarial example. The function $sgn(.)$ returns the sign of its argument. While larger ϵ will cause the neural network to misclassify more frequently, the perturbations will be much more noticeable as ϵ is directly proportional to the magnitude of the perturbation. Figure 2.2 shows an example of the FGSM attack.

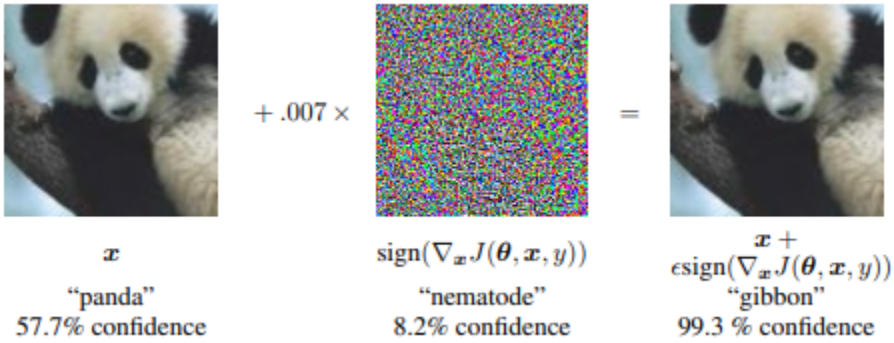


Figure 2.2: Small perturbation causing misclassification using FGSM attack [1].

The images of the panda on the left and right on Figure 2.2 are virtually indistinguishable. From an attacker's perspective, it is also important to find a solid middle ground for the strength of the perturbation, as too high of the strength makes the adversarial example too obvious, as shown in Figure 2.3.

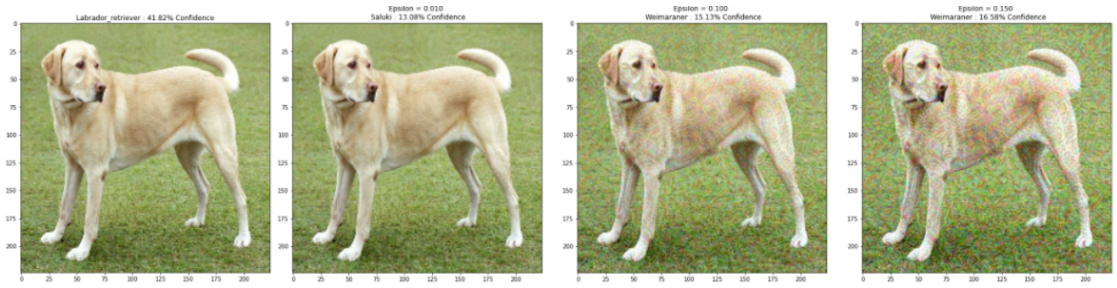


Figure 2.3: Effects of the strength of the FGSM attack [1].

As the strength (ϵ) increases, the confidence of the neural network misclassification increases, but the images are noticeably distorted. In Figure 2.3, the smaller of the strengths, 0.01, is preferred since the neural network already has misclassified the labrador as a saluki with minimal perturbation.

Other variations on the FGSM attack have been explored over the years. A typical FGSM attack performs a single modification of the image. Goodfellow et al. has extended the fast method by decreasing the step size of the attack and performing it iteratively [13]. This method is known as the Basic Iterative Method (BIM). Instead of using the full effect of the perturbation factor ϵ, the factor is divided down by the number of iterations to control to ensure the adversarial sample is within a specific distance from the original. FGSM, by nature, is an untargeted attack, where the perturbed image can be misclassified as any output in the domain. The targeted FGSM attack is a variant of the iterative attack where in addition to the standard generation of an adversarial example, the classification of the sample must also be of a specific class [13] [12]. The simplest form of the targeted FGSM perturbs the sample until the desired label is produced as an output.

Several defenses against adversarial samples have been developed over time. Good-

fellow et al. proposed the idea of adversarial training, where the loss function used to train the network was modified [1]:

$$\widetilde{J}(\theta, x, y) = \alpha J(\theta, x, y) + (1 - \alpha) J(\theta, x + \epsilon \times sgn(\Delta_x J(\theta, x, y))) \tag{2.2}$$

where x is the input, y is the output, θ are the parameters of the model, $J(\theta, x, y)$ is the loss function of the model, $J(\theta, x + \epsilon \times sgn(\Delta_x J(\theta, x, y)))$ is the loss obtained through FGSM, and α is the weighting factor. Goodfellow et al. determined that an α of 0.5 worked well for adversarial training. With adversarial training, the error rate from Goodfellow's model went from a 89.4% error rate to a 17.9% error rate, which is a significant improvement.

2.2.2 Model Extraction

Model extraction is the attempt at replication of a target neural network, or oracle. This can either refer to just duplicating the functionality of the oracle, or the copying the actual parameters of the network [6]. Most basic model extraction attacks are blackbox to an extent, as long as the attacker has access to the input and output of the oracle.

Several issues arise when considering a model extraction against a neural network. One of the most challenging issues is that the attacker will not know the overall network structure, so reconstructing a substitute network may be difficult. Secondly, training data may not be readily available, making it difficult to replicate the oracle's functionality. The querying of the oracle may be difficult, as the attacker may be limited to just random data, which can render the input-output pairs to be less useful. Lastly, if training data is available, many accesses to the input, or queries, may be necessary for model extraction. In the case of model extraction, the aforementioned problems may combine and require a much more complex attack [2] [5] [6].

2.2.3 Applications of Model Extraction and Adversarial Attacks

The authors of [5] performed a learning-based model extraction by observing the network's response to a number of inputs. A synthetic dataset is generated with the appropriate input size, and the trained oracle was queried with it to obtain a labelled dataset. For larger networks, generating all sets of inputs would exponentially increase the number of queries. To limit the number of synthetic points and generate meaningful data, the authors use a method called Jacobian-based Dataset Augmentation to create more synthetic training points. This is done by evaluating the substitute neural network's Jacobian matrix on the input-output pair generated by the oracle:

$$x_{aug} = x + \lambda \times sgn(J_F[O(x)]) \tag{2.3}$$

where x_{aug} is the new datapoint, x is the original input, λ is a parameter of the augmentation, O is the oracle, , J_F is the Jacobian matrix, and $O(x)$ is the output label. The new points are generated from a small set that encompasses all possible members of the output domain. For example, for MNIST, the smaller set would consist of at least one of each handwritten digit. After obtaining the the available samples in the dataset, an network architecture appropriate for the task is chosen. During the training of the surrogate, the Jacobian dataset augmentation is used to create new samples. The trained surrogate can then be used to generate adversarial examples that the oracle will misclassify. This model extraction, while it produces results close to the oracle, does not necessarily replicate the oracle architecture, and thus, does not replicate the parameters. However, it is definitely quicker because less queries need to be performed on the oracle overall, as the number of initial samples can be small. In their model extraction for an MNIST model, the adversary only had to perform 6,400 total queries instead of 50,000. While the accuracy wasn't up

as high as the state of the art, the overarching goal was to produce a model that would learn the oracle's decision boundaries rather than a substitute model with high accuracy.

The authors of [6] extend upon the work of Papernot et al. [5] and introduce a second type of attack that aims to retrieve the functionally equivalent model. They use two metrics to evaluate the success of the attack - *accuracy*, or how well the model performs on the original labelled dataset, and *fidelity*, which measures the accuracy of the outputs of the extracted model and oracle. A high-fidelity surrogate, for example, would even replicate the errors produced by the oracle. The number of queries needed and the difficulty in proving the functional equivalence of two networks are highlighted as the main limitations as the complexity of the networks grows. For the functional based extraction, they target a neural network with a single hidden layer with the Rectified Linear Unit (ReLU) activation function. The attacker is assumed to have access to the input, output, and outputs of the activation function of the network. It is also assumed that the attacker knows the activation function to be ReLU. They exploit the fact that the ReLU is a piecewise function to help extract the weights of a layer. Input(s) are identified such that exactly one of the ReLU units outputs a 0 (also called a critical point). These are determined sweeping randomly sampled vectors in the function:

$$L(t; u, v, O_L) = O_L(u + tv) \tag{2.4}$$

where $L(\cdot)$ is a piecewise linear function, u, v are the randomly sampled vectors, t is the location in the function where it is not differentiable (critical point) and O_L is the logit function:

$$O_L(x) = A^{(1)} ReLU(A^{(0)} x + B^{(0)}) + B^{(1)} \tag{2.5}$$

where x is the input, $A^{(0)}$ and $A^{(1)}$ are the hidden and output layer weight matrices, and $B^{(0)}$ and $B^{(1)}$ are the biases in the hidden and output layers. Once t is determined, the input is then used to find the difference between the two regions created by the critical point to recover the row weights. After applying it to each ReLU in the layer, the weight matrix is obtained. After the sign of the weights are recovered, the weight matrix from the hidden layer to the output layer is solved using least-squares. The authors of [14] perform a similar attack by targeting the ReLU activation critical points but perform the attack on deeper neural networks by solving the weight matrices one hidden layer at a time. While both of these model extraction attacks are extremely effective, the method is computationally intensive, especially for larger networks. It also requires access to the activation function of the network, which isn't always accessible to an attacker. In addition, the only activation function the attack model works against is the ReLU activation function, as it has a noticeable critical point.

Troung et al. pioneered a method where a model extraction attack was performed against an oracle where little to no training data was available to train the surrogate model [2]. In a typical model extraction attack, an adversary typically has access to some parts of the dataset used to train or test the oracle model, and may be able to generate more datapoints from the available samples. The adversary uses these samples to query the oracle to obtain a labelled dataset to train the surrogate. For more proprietary models, these training and test samples are hard to acquire. The authors propose a technique called data-free model extraction (DFME). A setup inspired by a Generative Adversarial Network (GAN) is used, where a generator attempts to create realistic samples to fool the discriminator. In traditional GANs, the generator is trained until the data distribution matches the target data, and the discriminator is unable to differentiate between an actual and generated sample. During the training of a GAN, the gradients backpropagate through the discriminator

through to the generator, but the weights of the discriminator are not modified. After the generator is fully trained, the discriminator is usually discarded, and the generator is used for a variety of tasks, such as sample generation and data augmentation. In the proposed DFME, the discriminator is replaced by the student (surrogate) model as shown in Figure 2.4. The victim (oracle) model is used to calculate the loss.

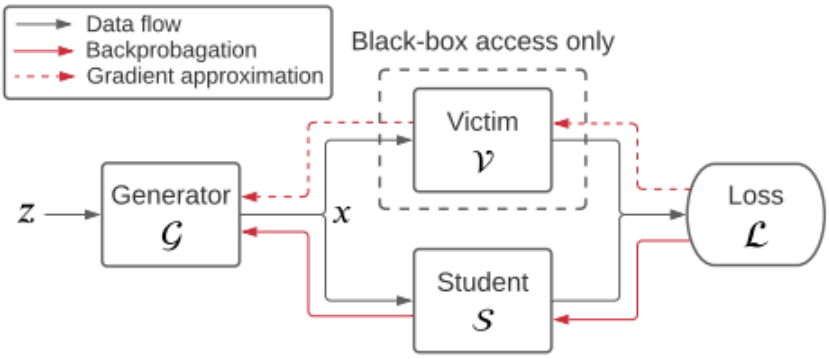


Figure 2.4: GAN inspired DFME diagram [2].

The generator (G) creates input images (X) that are passed to the student (S) and victim (V) models. The student model acts as the discriminator in this setup as it attempts to learn the predictions of the victim model. Both models produce an output that is used in the custom loss function. After the loss is determined, two separate gradients are calculated - the gradient with respect to the student parameters, and the gradient with respect to the parameters in the generator. The generator and the student are trained on the same loss function, but on separate ends of the spectrum. The student attempts to learn the predictions of the victim, while the generator attempts to create examples that the student will have difficulty in classifying. In other words, the generator attempts to maximize the loss function, while the generator attempts to minimize it. The authors tested their setup on two datasets - SVHN and CIFAR-10. The ResNet-18-8x architecture was chosen for the student, and a 3-layer convolutional network with up-sampling layers, batch normalization, and ReLU activations (besides the last layer, which uses the hyperbolic tangent to constrain the

output to ensure the victim accepts the input image). The student was trained using stochastic gradient descent (SGD) with a batch size of 256. The generator was trained with the Adam optimizer and a batch size of 256. Both networks were trained with learning rate decay at various points of training. A query limit for the victim model was also used to observe the change in accuracy. For CIFAR-10, the student was able to achieve an accuracy of 88.1% with 20 million queries, and 89.9% with 30 million, compared to the target of 95.6% achieved on the victim. For SVHN, the student was able achieve 95.2% accuracy with just 2 million queries, which is impressive as the victim's accuracy was 96.2%. Similar to previous model extraction attacks, the goal of the surrogate model is to attempt to mimic the functionality of the oracle, rather than attempt to recreate the model itself. In addition, an adversary must select a network with the appropriate size and structure for the task, which may be difficult as the oracle is completely blackbox. However, this may be circumvented by knowing the general task the oracle is performing (such as regression, object recognition, etc.). The main advantage of this model extraction attack is the ability to train a surrogate with as little as random noise from a normal distribution.

A majority of model extraction attacks are performed on MLPs due to their simple architecture style, and CNNs, as they are used in a variety of tasks. Model extraction attacks have begun to branch out to other styles of neurasl networks. Takemura et al. performed a extraction attack on recurrent neural networks (RNNs), which excel at handling time-series data [15]. They observed that a model with higher accuracy can be obtained by choosing a complex substitute model and loss function.

2.3 Side-Channel Attacks & Neural Networks

Side-channel attacks (SCA) are a set of attacks that use observable results of a system rather than faults of its implementation [16]. Many side-channel attacks are performed on blackbox models, where the actual implementation is obfuscated. Common ex-

amples of side-channel information include measuring power consumption, analyzing timing between inputs and outputs, observing emitted sound, and checking memory accesses [16]. They are mainly used in the field of cryptography, usually to decipher and retrieve keys of encryption algorithms such as the Advanced Encryption Standard (AES) or to obtain the state of the system. In addition, SCAs are known for reducing the complexity of attacks, as most of them split the attack up into multiple smaller parts [4]. Although side-channel attacks are commonly used in the field of cryptoanalysis, some SCAs have been used in neural networks to recover input information and the model parameters themselves. One example of an SCA is on hardware-based neural networks, where some side-channel information, such as power consumption and timing, are easier to obtain and analyze.

A binarized convolutional neural network (BCNN) was developed by Courbariaux et al. [7], and optimized by the authors of [17], who moved the convolution unit into a unit called the line buffer. The authors of [3] noticed that the line buffer in Zhang et al.'s design had the highest power consumption. Wei et al. noted two possible attacks - an active and passive attack, depending on the level of access the attacker had on the network. Both styles of attack attempted to recover the input image into the network using different types of power analysis. The inputs used for the attack were images from MNIST dataset. This dataset features white, handwritten digits from 0 to 9 on a black background.

In a passive attack, differential power analysis is used to recover the background of the image. Differential power analysis can be defined as observing the difference in power consumption between subsequent inputs. As MNIST features white numbers on a black background, most of the pixels in the background are of similar value. When used as inputs into the network, similar pixels generally leads to similar power consumption between them. Thus, a series of power consumption values for pixels that are around the same value implies that those pixels are all in the foreground

or the background. For the passive attack, lower power consumption implies darker pixels, which is the background for the MNIST dataset. By revealing the background of the image, the foreground is also revealed. However, no details outside of the shape of the foreground can be revealed in a passive attack. A threshold is determined by finding the differential power consumption between the two power trace cycles that are the largest. A filter is applied to the pixels above the threshold, and the remaining pixels are the background. An example of the passive attack can be seen in Figure 2.5.

Figure 2.5: Input image recovery using differential power analysis - background detection [3].

The active attack assumes that the attacker has a higher level of access to the model, and is considered more of a whitebox attack. This attack reveals the background of the images as well as more detail in the foreground compared to the passive method. Instead of differential power analysis, the authors use a power template to attempt to obtain the pixel values. A power template is the collection between pixel values and their power consumption. Access is required to the line buffer, as the power template gathers data from the power consumption by observing the convolution operation. Since each pixel in MNIST can range from a value between 0 and 255, the search space for the pixel values is rather large, and brute forcing the attack is not a viable option. Since the convolution operation shifts from left to right and from top to bottom, and the kernel size of the convolution remains the same throughout the convolution operation, the the differential power between each convolutional operation can be observed to obtain potential pixel values for that cycle. To produce a final pixel value, a pixel value is selected from each cycle that produces the least

overall variance. The pixel values are then averaged to produce a final value. An example of the active attack can be seen in Figure 2.6.

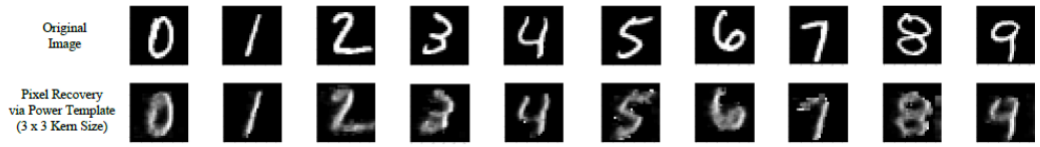


Figure 2.6: Input image recovery using differential power analysis - power template [3].

There are several limitations relating to this work. For the passive attack, the authors relied on the input images having a distinct foreground and background. While this is easily doable on a dataset like MNIST, more complex datasets may have non-uniform background and multiple objects in the foreground, making it difficult to perform any differential power analysis. However, the passive attack is relatively simple and is also a blackbox attack, as it relies on observing and analyzing the power consumption of the model, making it more accessible. The active attack, on the other hand, can potentially be effective on more complex images. However, as it is a whitebox attack, it can be harder to accomplish. For both attacks, it is also unclear whether the same attacks can work on other networks, such as non-binarized networks.

Some work has been done on using side-channel information to recover neural network parameters from hardware. The authors in [18] mounted a model extraction attack on a DNN accelerator implemented on an Field Programmable Gate Array (FPGA). An small MLP (single hidden layer with 5 neurons) was implemented on the FPGA, with the parameters (in 8-bit fixed point format) stored on off-chip memory. The majority of the calculations performed in the multiplier and the accumulator register. The adversary would use correlation electromagnetic analysis to relate the electromagnetic leakage and the hamming distance of the intermediate calculations that are stored in the accumulator. Preliminary results showed that the adversary was able to obtain 95% of the weights through 60000 measurements made. While it

is impressive that the weights were successfully recovered, several other factors can be considered in potentially improving the attack. As the authors state, one shortcoming is that the model being attacked is relatively simple in nature, as there are only 20 weights to consider. Adding more neurons and layers could exponentially increase the complexity of the attack, as more leakage measurements would be needed. The parameters for a hardware neural network could also be stored in floating point format, which could complicate the correlation electromagnetic analysis, as the hamming distance cannot be as easily obtained. There are concerns that side-channel measurements, especially those made on off-chip memory, may be noisy and will not provide an accurate reconstruction of a network. This is a valid concern, as there can be millions of parameters to extract between the weights and the layers of the network. Hu at el. showed that model extraction was possible with high accuracy using off-chip address traces and peripheral component interconnect express (PCIe) events from a typical GPU obtained "off the shelf" [19].

Similar work has been done with CNNs implemented on hardware accelerators. Hua et al. used timing and memory side-channel information to predict the architecture of the underlying network [20], even if the parameters of the network are stored off-chip. Weights can also be extracted, but the two types of attacks require different levels of access to the model. It is assumed that the feature maps of various layers are stored in memory. To reveal the structure, visibility of the read and write access to memory and training data are necessary, while weight extraction requires input manipulation and access, knowledge of the network structure, and write accesses to memory need to be visible. To obtain the network structure, the authors exploit read-after-write (RAW) accesses, which can be viewed by observing a read after a write on the same memory address. The boundary before a convolutional or fully connected layer can then be determined by observing the first read access on a memory address after a write, as these are the only layers that modify feature maps. Other layer

parameters are then determined by observing other access patterns. In any given layer, there exists an input feature map and and output feature map, which are also written into memory. The output feature map is typically only written into once. The input feature map can be determined by observing a read in a memory address written in the previous layer. Filters in the network are read-only. As these are stored in contiguous parts of memory, the sizes of the input and output feature maps and filters can be determined using access information between layers. The specific dimensions of these parameters can then be narrowed down by determining the number of multiply and accumulate (MAC) operations for that layer. To reverse-engineer the weights, the same idea proposed by Jagielski et al. [6] where the critical point in the ReLU activation function is exploited after the model structure is determined. Hua et al.'s model extraction attack is effective, as limited knowledge about the model is needed to learn about the layer parameters. However, for more proprietary CNN models, training data may not readily available to the adversary. Without training data, read and write accesses may not provide the correct information needed to determine the network architecture, meaning the weights of the network may not be readily determined.

The authors in [4] were able to use side-channel information to reverse engineer a neural network implemented on a microcontroller. Aspects of the neural network that could be obtained using this method are activation functions, size and number of the hidden layers, output classes, and weights in the neural network. For the attack, the authors assume little about the network. One assumption is that the network being targeted is an MLP, as it is simpler to work with and is still used in modern machine learning applications. In addition, the attacker has access to the input of the network, but does not know the parameters of the network. The authors pre-compile and train the network off-chip.

To recover the activation function, the authors measure the timing delay of various

activation functions with identical inputs through an electromagnetic trace, as shown in Figure 2.7.

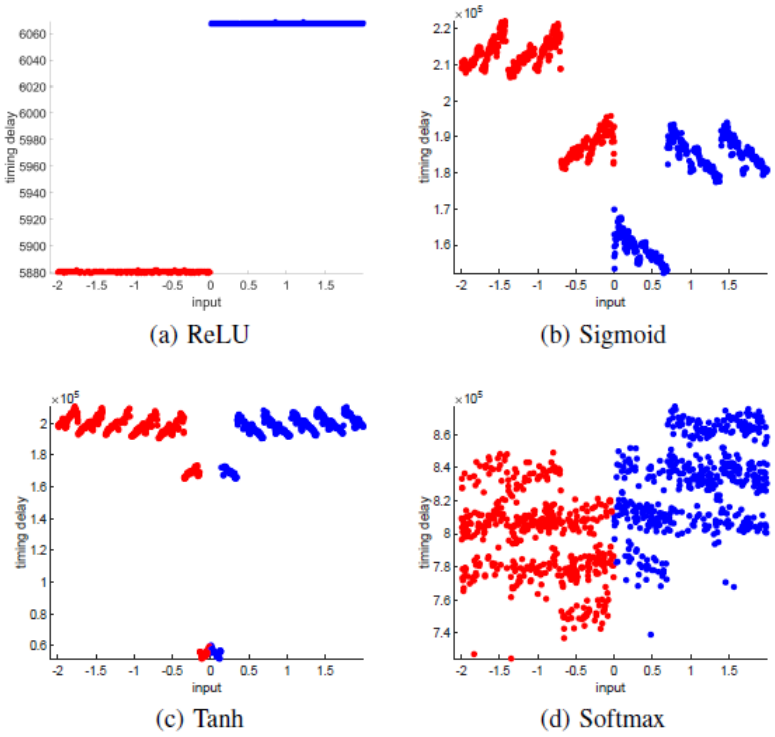


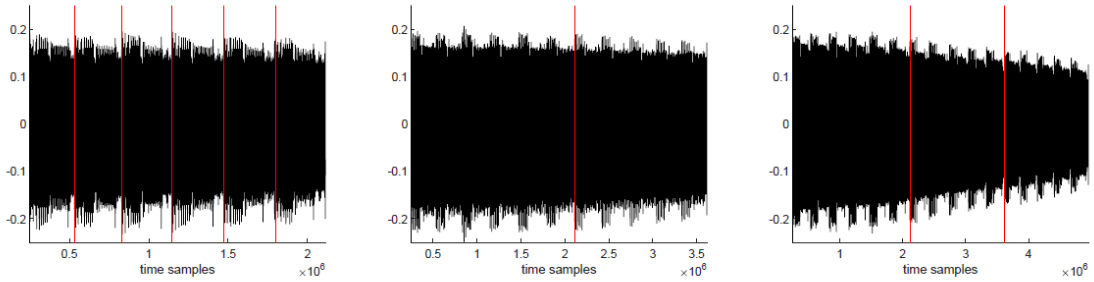
Figure 2.7: Timing delay of commonly used activation functions [4].

The different activation functions produce different timing delays, with the order of complexity going in order from ReLU, tanh, sigmoid, and softmax. tanh and sigmoid both require an computation of e^x, while softmax requires exponentiation and is dependant on the size of the output layer. Because the timing delays of each of these activation functions is visibly unique, it is easy to determine the activation function being used.

To recover the weights, the authors used Correlation Power Analysis (CPA), which is differential power analysis with the addition of using the Pearson's correlation as an additional test. They use CPA to target the multiplication function between a known input and a hidden weight. The adversary correlates the output of the multiplication with the side-channel measurement for all possible values of the weight. The Pearson

correlation coefficient is then calculated between them, which can narrow the search space - the higher the correlation, the more likely the weight is correct. The authors then recover the weight in three components. As the network is implemented on a microcontroller, the various parameters of the network are encoded as IEEE-754 floating point representation, and they recovered the sign, mantissa, and exponent separately. CPA was run on all three components to obtain a unique weight.

To recover layer size and number of layers, the activation and weights must have been reverse engineered first. The authors then use the most basic SCA, Simple Power Analysis (SPA) to determine the rest of the parameters of the network. SPA is the analysis of an implementation uses very few traces, compared to differential power analysis (which may need hundreds of thousands of measurements). Figure 2.8 shows the extraction of the number of layers as well as the number of neurons in each layer.



(a) One hidden layer with 6 neurons (b) 2 hidden layers (6 and 5 neurons each) (c) 3 hidden layers (6,5,5 neurons each)

Figure 2.8: Neuron count and layer extraction [4].

The authors exploit the fact that the power signatures of the hidden layers are often different than the activation function. As seen in Figure 2.8, the different layers are clearly visible. In (a), the red lines indicate the different neurons in the single hidden layer, while in (b) and (c) the red lines are the activation functions. Determining the layer boundaries is slightly more complex, however. To do so, a weight recovery is done on two separate instances - one with the an unknown neuron in the current layer, and another the neuron in the next hidden layer. The weight recovery with the higher correlation value determines the location of the boundary.

While it is impressive the attacker is able to recover much of the neural network parameters, there are a few setbacks. Recently, there has been uptick of a variety of new architectures that can perform modern day tasks much more efficiently. Since this model extraction is limited to an MLP, it is unclear if the attack is easily transferable from one architecture to the next. The transferability of this attack to other hardware, such as an FPGA, GPU, or other microcontrollers is also unclear. In addition, most modern networks are rather large, which can increase the time needed to execute this attack. This attack also requires very specific hardware, which may not be widely available to an adversary. One improvement that can be made is to create a generic model that can eventually be adapted to various piece of hardware for increased transferability.

Chapter 3

Model Extraction and Blackbox Threat Model

The first objective of this thesis was to explore the effect of surrogate training set size on the fidelity of model extraction and transferability of FGSM attacks when no side-channel information is used. Figure 3.1 shows the threat model used for the attacks.

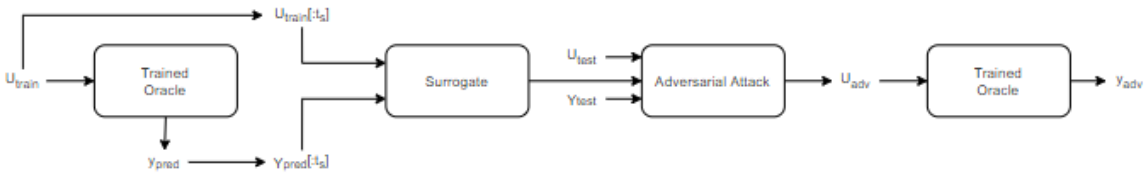


Figure 3.1: Block diagram of threat model.

First, the trained oracle is queried using the training set, U_{train} and produces the predictions y_{pred}. To train the surrogate model, a subset of the training set and predictions is used determined by the surrogate training set size t_s. Once trained, the surrogate parameters would be used in an adversarial attack along with samples from test set, denoted by U_{test} and Y_{test}, to generate adversarial examples U_{adv}. U_{adv} would be passed back into the trained oracle to get the adversarial predictions y_{adv}.

A binarized multi-layer perceptron (BMLP) with binarized activations was used as the oracle model. It was also assumed that the attacker has perfect knowledge of the oracle model (topology, activation function, etc.) except for the parameter values. The output layer used the softmax activation function. Figure 3.2 shows the

architecture for the oracle and surrogate models.

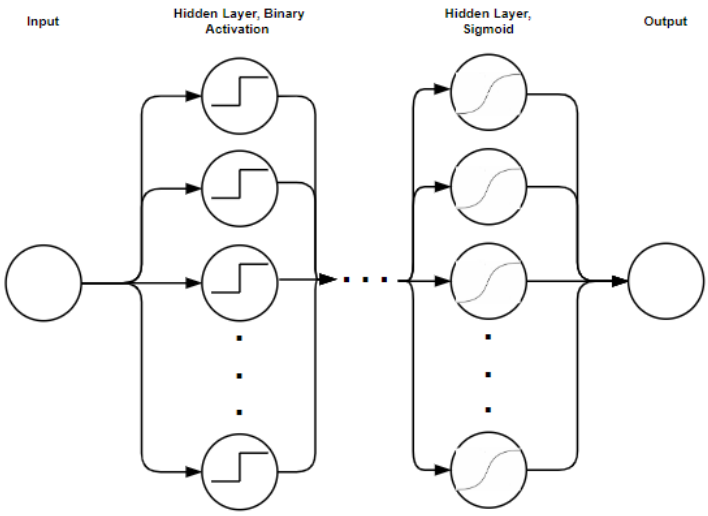


Figure 3.2: Multi-layer perceptron model with binary activation.

The binary activation, $b(x)$ is defined as:

$$b(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases} \tag{3.1}$$

A network with a single hidden layer was chosen to analyze. As the binary activation is not differentiable at the threshold (when the input is 0), a sigmoid approximation gradient was used to provide a smoother gradient:

$$\delta(x) = \frac{1}{e^{-2x}}(1 - \frac{1}{e^{-2x}}) \tag{3.2}$$

The MNIST dataset was used to train and test the oracle and surrogate models. The MNIST dataset consists of 70,000 black and white 28x28 images of handwritten digits from 0-9, split into 60,000 training examples and 10,000 testing examples. Figure 3.3 shows examples of the MNIST dataset.

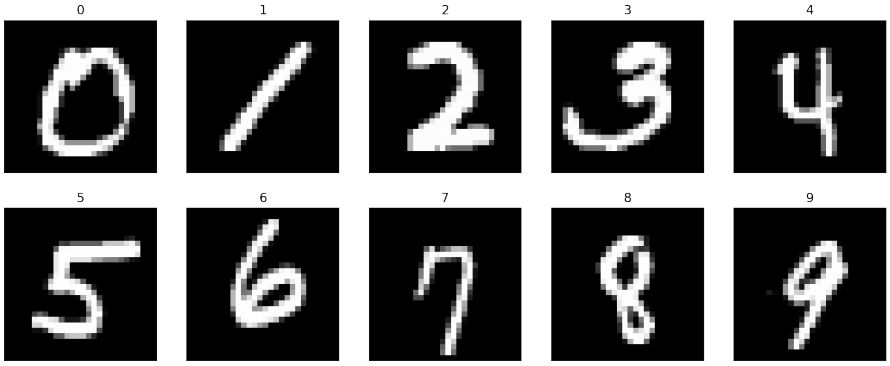


Figure 3.3: MNIST dataset image examples.

The hidden layer consisted of 100 neurons and the output layer consisted of 10 neurons. The numerical labels of the MNIST dataset were converted into one-hot encoding labels to use the softmax activation in the output layer.

3.1 Model Extraction

The oracle model was trained on the full MNIST training data, and was queried to obtain its predictions on the training set. The surrogate model was trained with different-sized subsets of the MNIST training data and the oracle predictions to determine the training set size's effect on model extraction. Training set sizes ranged from 1 to 60000. The surrogate model was identically initialized for each training set size. After the each training set size, the weights between the oracle and the surrogate were compared using mean-squared error (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(X_i - \hat{X}_i)^2 \tag{3.3}$$

where n is the total number of parameters, X_i are the vectorized weights matrices of the surrogate, and $\hat{X}_i$ are the vectorized weights matrices of the oracle.

The surrogate was trained on each of the training set sizes 50 times, and the results were averaged. A run consisted of the training of the oracle and each training

set size for the surrogate. While the weights before the training of the surrogate were identically initialized during each run, they were not identically initialized between runs. Figure 3.4 shows the relationship between the number of training samples used to train the surrogate and the MSE of the weight matrices.

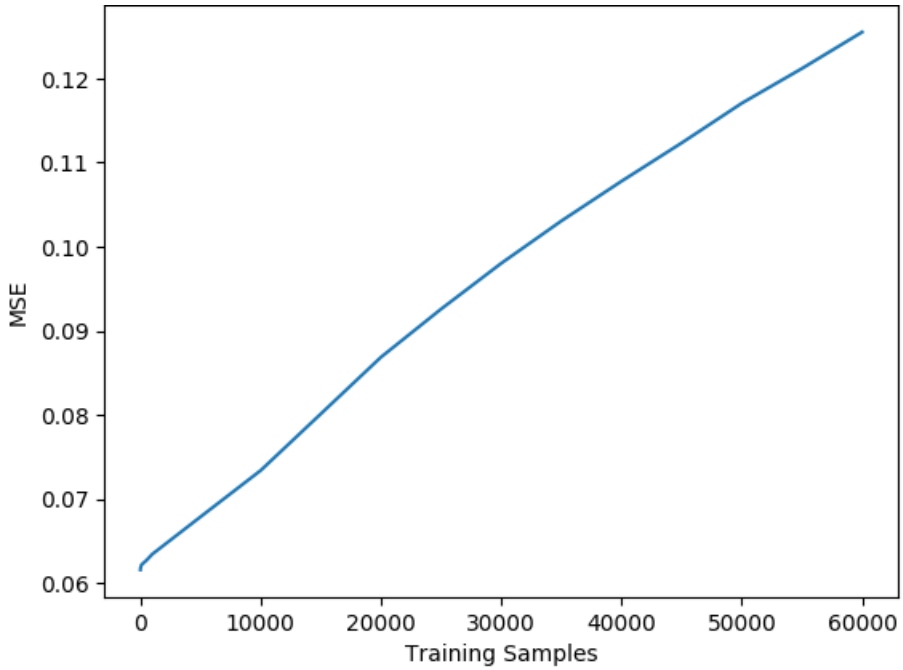


Figure 3.4: Mean-squared error of the weights between the oracle and the surrogate.

The MSEs between the various training sample numbers used to train the oracle grows linearly as the number of training examples increases. It was to be expected that as the number of training examples increases, the MSE would decrease. One possible reason why the MSE increased could be because of the initializations of the models. With fewer training samples, the surrogate model did not have as many weight updates, and thus did not deviate as much from its initialization. With more training examples, there were significant weight updates, driving the distance of the weights further apart.

Another metric, the angle between the weight matrices, was computed to measure

how close the weights between the oracle and surrogate were using:

$$\theta = \arccos\left(\frac{\mathbf{o} \cdot \mathbf{s}}{|\mathbf{o}||\mathbf{s}|}\right) \tag{3.4}$$

$\mathbf{o}$ is a flattened vector of the weights in the oracle and $\vec{s}$ is a flattened vector of the weights in the surrogate. The unit vector of each flattened vector was found by dividing by the magnitude, and the dot product was taken. Figure 3.5 shows the angle between the weight matrices.

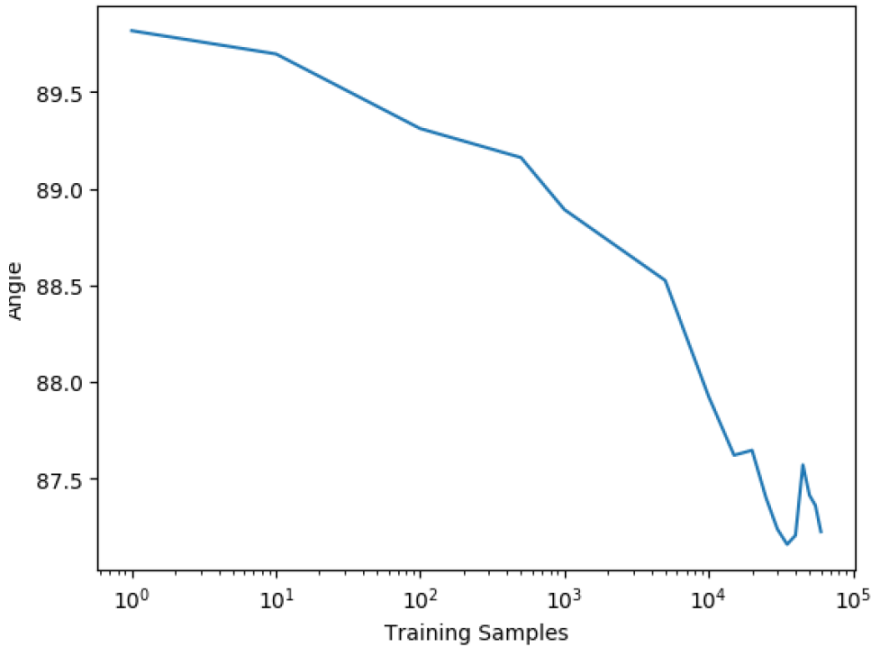


Figure 3.5: Angle between the weights between the oracle and the surrogate.

The angle between each of the weight matrices for each training sample seems to be roughly orthogonal. However, increased training set sizes does cause the angle to decrease linearly by a very small amount. This could also be the result of the initializations of the model, where the vectors of the oracle and surrogate are moving in the same direction away from each other, resulting in similar angle measurements. The slight decrease could simply mean the surrogate is starting to slightly move towards the oracle model.

3.2 Attack Transferability

A series of adversarial attacks, more specifically FGSM attacks, were mounted on the oracle and surrogate models trained on MNIST. First, a whitebox attack was performed on the trained oracle using the 10,000 test samples to obtain a set of adversarial images. A whitebox attack was also performed on the trained surrogates, regardless of the number of training samples used to train it. Two relative accuracies were calculated - one for the whitebox attack on the oracle, and a blackbox attack on the oracle, where the adversarial images from the surrogate were used to attack the oracle. A relative accuracy is defined as the accuracy of the model on the adversarial examples divided by the accuracy of the model on the unperturbed images. This metric allows for the comparison of how strong the blackbox attack is compared to the whitebox attack.

Several values for the strength, ϵ, were used to test the effects of the scaling on the attack. Values used for ϵ ranged from 0 to 1. More ϵ values that were tested were between 0.1 and 1, as lower epsilon values did not add enough noise to the adversarial image to cause a large number of misclassifications.

A sample adversarial image and its added noise for each surrogate and each ϵ was generated, as shown in Figures 3.6 and 3.7 to verify the FGSM implementation.

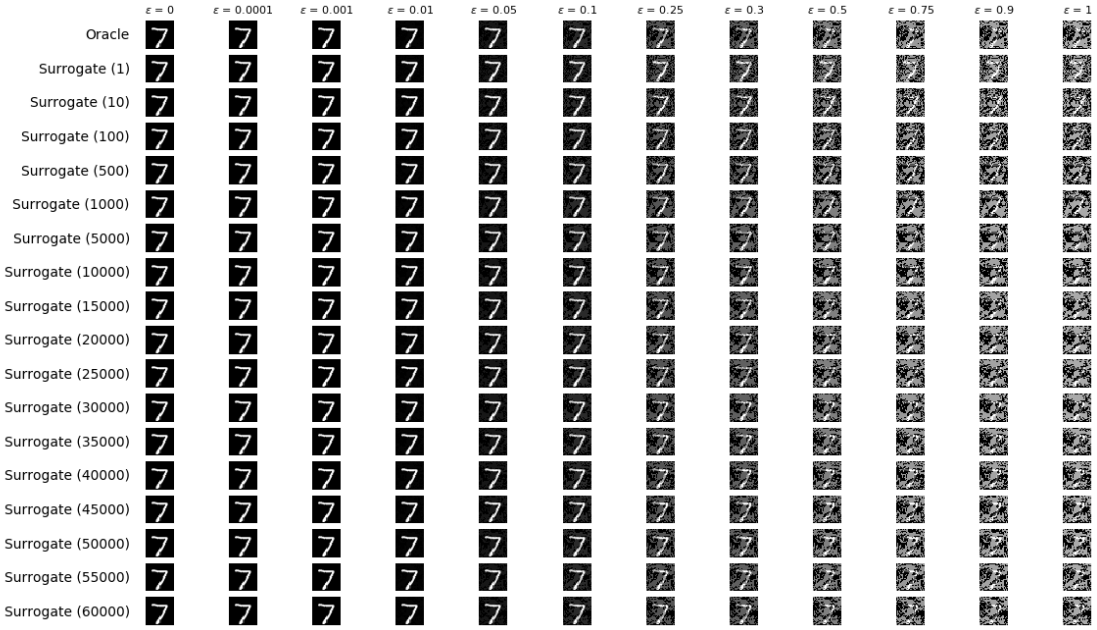


Figure 3.6: Sample adversarial image generated by FGSM.

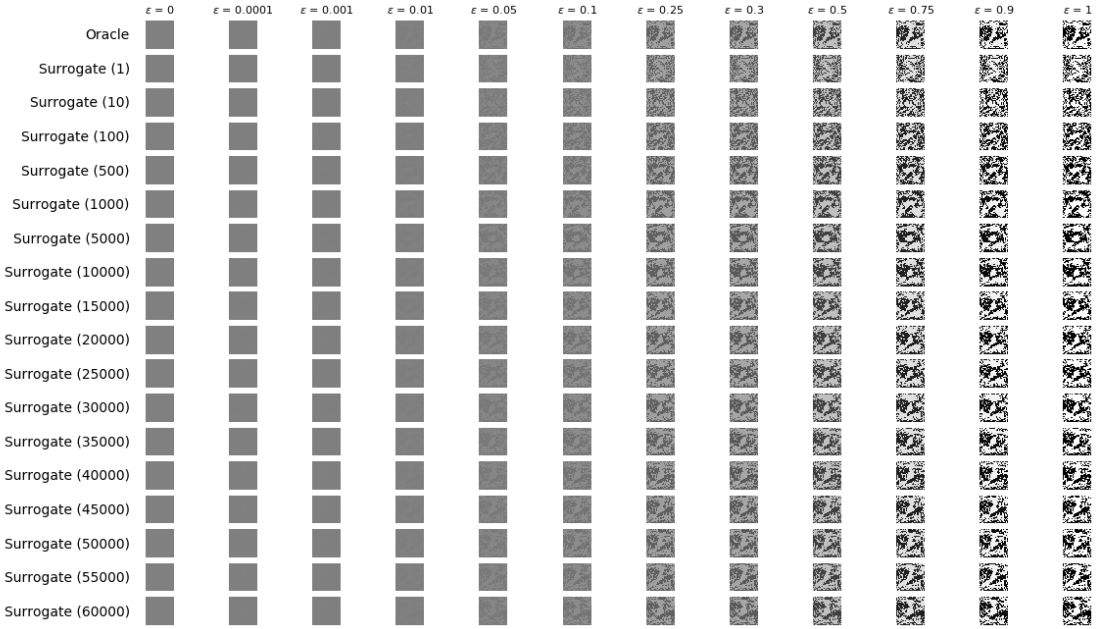


Figure 3.7: Sample noise added to images by FGSM.

Besides for the case of $\epsilon = 0$ (as no noise was added), the adversarial image generated by the surrogate was closer when more training samples were used to train the surrogate, which is expected as the functionality of the highly trained surrogate

would be closer, as shown by the higher test accuracy. Higher perturbation factors result in samples that are visibly more distorted, as the added noise is quite large.

Figure 3.8 shows the average relative accuracy plot of the whitebox attack on the oracle vs. the blackbox attack on the oracle.

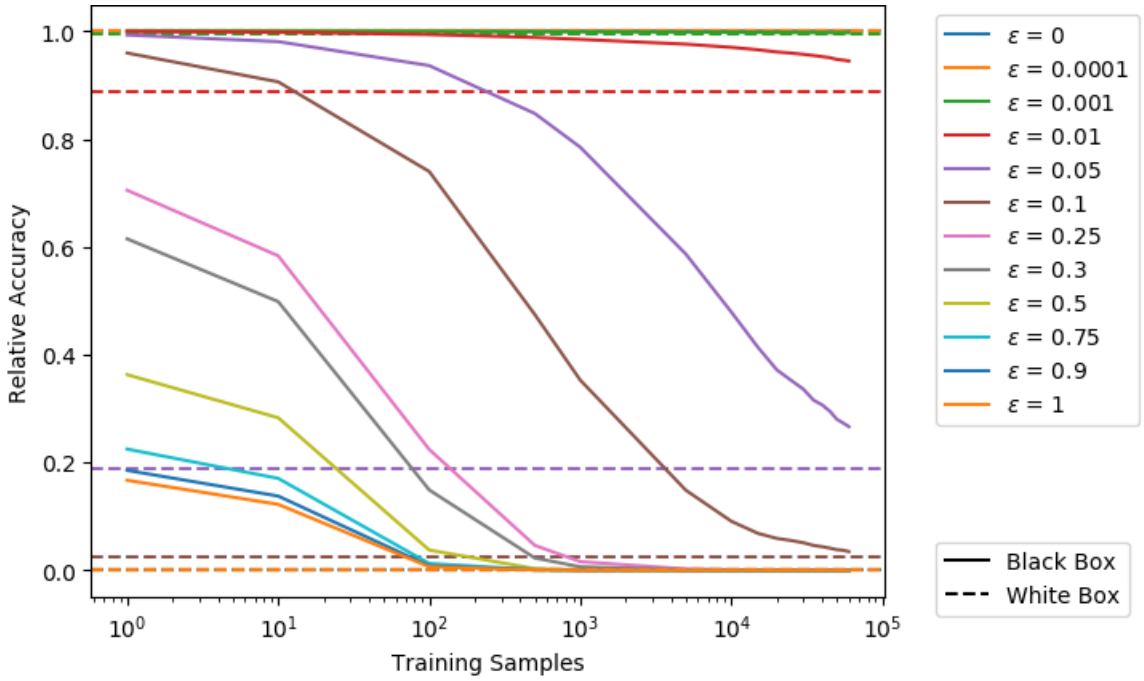


Figure 3.8: Whitebox vs. blackbox relative accuracies.

At lower epsilon values, the relative accuracies for both sets of attacks are very close to 1.0, as not much noise was added to the image. The first noticeable change in relative accuracy occurs at $\epsilon = 0.01$. For all ϵ, the relative accuracy from the blackbox attack asymptotically approaches the relative accuracy from the whitebox attack. This is expected, as the differing weights between the oracle and surrogate would produce different gradients, and thus, different perturbations would be generated in the attack. Because the weights of the oracle and surrogate are not equivalent, there exists a possibility that the surrogate may perform differently.

Next, the transferability of the attack was considered. The transferability of the attack is defined as the percentage of attacks that were successful on the surrogate

that were also successful on the oracle:

$$T = \frac{A_{oracle} - A_{0,oracle}}{A_{surr} - A_{0,surr}} \tag{3.5}$$

where A_{oracle} and A_{surr} are the number of adversarial samples that caused a misclassification on the oracle and surrogate, respectively, and $A_{0,oracle}$ and $A_{0,surr}$ are the number of adversarial examples that caused a misclassification on the oracle and surrogate, respectively but were not classified correctly with the original image.

For both cases, attacks only count for the transferability metric if the unperturbed sample was correctly classified in both the oracle and surrogate (or at $\epsilon = 0$). Figure 3.9 shows the transferability of attacks from the surrogate to the oracle.

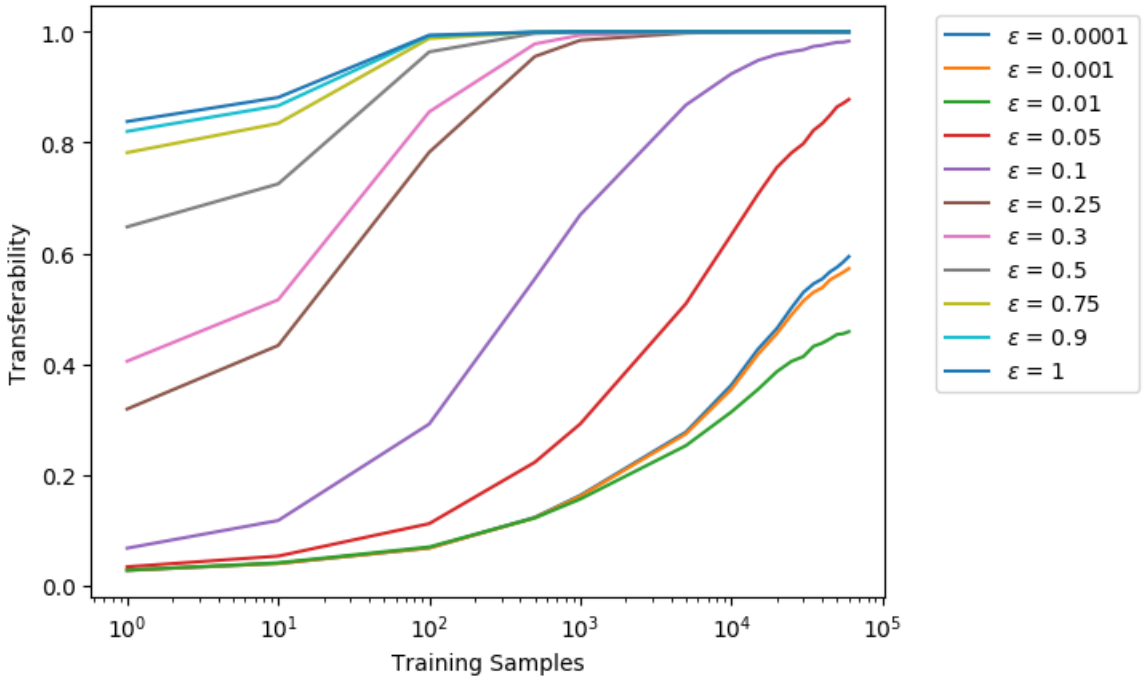


Figure 3.9: Transferability of the FGSM attack.

At lower epsilon values, attacks are less likely to transfer, as the adversarial images are unlikely to cause either the oracle or surrogate to mispredict. As expected, more training samples result in higher transferability, as the functionality between the two networks is closer. As ϵ increases, surrogates trained on fewer training examples are

more likely to have attacks transferred, as there was increased noise being added to the image.

Next, a breakdown of the attacks was generate to help visualize the effects of the attacks on the oracle and the surrogate. The data was split into four parts - attacks that only affected the oracle, attacks that only affected the surrogate, attacks that affected both networks, and attacks that affected neither. Figure 3.10 and 3.11 show the attack breakdowns for various ϵ values. The test accuracies of the oracle and surrogate were also plotted for reference.

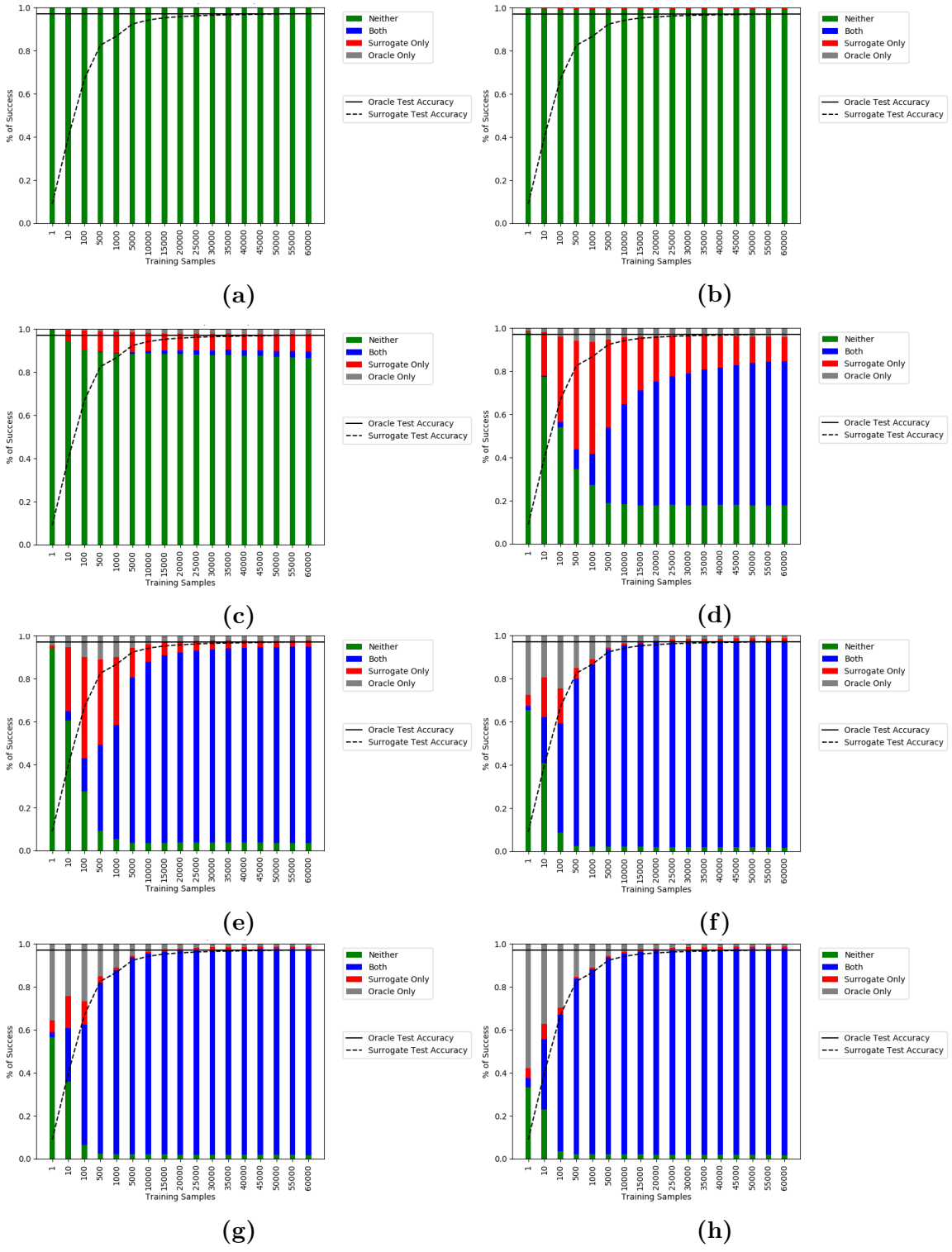


Figure 3.10: Attack breakdown for various ϵ values.

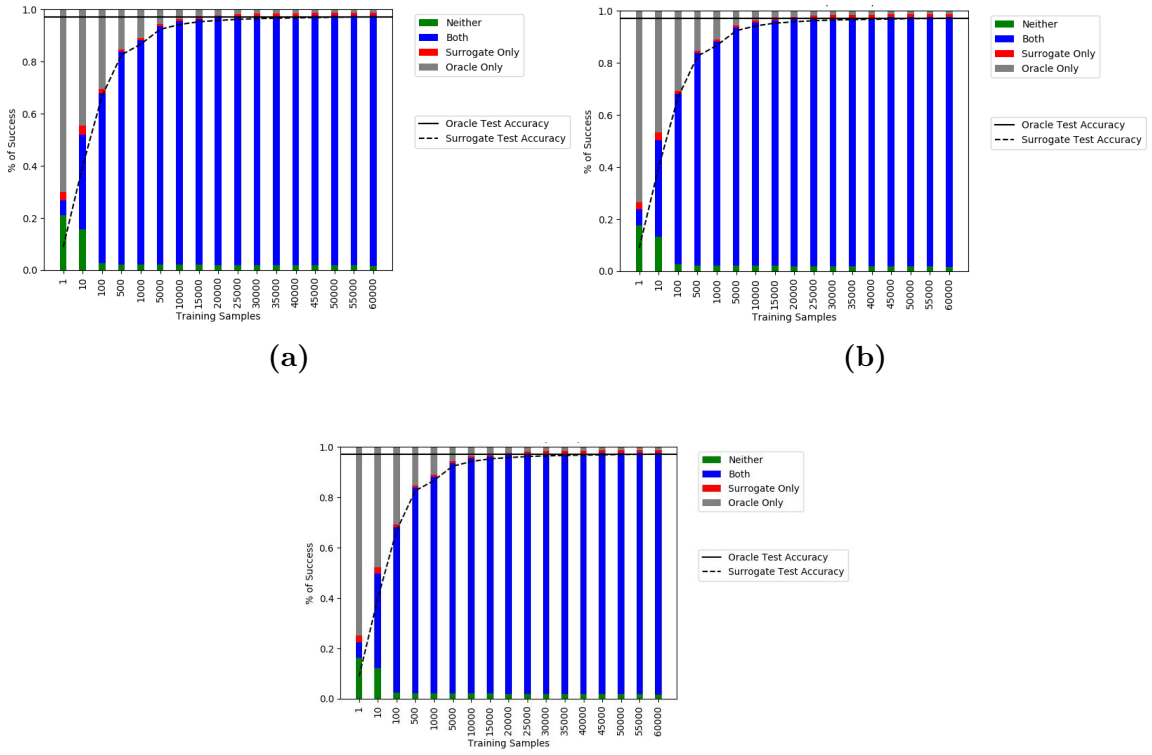


Figure 3.11: Attack breakdown for various ϵ values.

At lower ϵ values, the breakdowns are mainly filled with attacks that affected neither the oracle and the surrogate. As it increases, higher numbers of attacks where both networks are affected increases. In addition, the number of attacks where the oracle only is affected increases, which is likely due to the drastic difference between network functionality at lower training samples, since the surrogate model was not trained for all output classes. Thus the gradient calculated for the FGSM attack will be quite different, which could leads to higher oracle misclassifications. These samples did not affect the surrogate, as the definition of a successful attack is when an image is initially classified correctly but misclassified once perturbed. As the test accuracy for these surrogate models were lower, they often misclassified the clean samples. An interesting observation is that was ϵ increases, the summation of the percentages of attacks where both or neither networks were affected seems to be approximate to the

curve of the test accuracy of the surrogate.

Chapter 4

Power Model Creation

4.1 Power Theory

The second objective of this thesis was to explore how side-channel information could be used to improve model extraction. One possible source of side-channel information the measurement of the power consumption of a given model. The power consumption of a CMOS circuit consists of two main components - the dynamic power and the static power. Dynamic power is primarily the power consumption when switching activity, defined as a change of state from low to high (0 to 1) or high to low (1 to 0), is present at nodes of the circuit. Dynamic power consumption also includes short-circuit power and power consumption resulting from hazards from the circuit. Short-circuit power dissipation occurs when the pull-up network and pull-down network of a CMOS circuit are on during switching. Short-circuit power can be significant if the time it takes to transition input states is high - otherwise it is usually low compared to the switching power. Hazards in a circuit will cause certain nodes in the circuit to undergo glitching, which is a unwanted, sudden transition of state at a node before it settles into its intended state, leading to increased power consumption. Static power, conversely, is the power consumption observed when the circuit is idling. Equation

4.1 shows the calculation of the total power consumption of a CMOS transistor [21].

$$P_{total} = P_{switching} + P_{short\_circuit} + P_{glitch} + P_{static} \tag{4.1}$$

In an ideal setting, where the inputs transition quickly and no hazards are present, the dynamic power consumption would mainly consist of the switching power. In a complementary CMOS logic gate, where there is a pull up network (PUN) and a pull down network (PDN), the switching power is greater during the transition between 0 to 1 rather than 1 to 0. When the input into the CMOS gate is 0, the PMOS in the PUN is on, allowing current to flow from the power supply and begins to charge the output capacitance. When the input is then changed to 1, the NMOS in the PDN turns on and the PMOS turns off, cutting off the current from the power supply, and causing the output capacitance to begin to discharge. Charging the output capacitance leads to the largest overall switching power consumption, which occurs only on transitions from low to high [21].

The following equation shows the various components used to calculate the switching power of a CMOS circuit:

$$P_{dynamic} = \alpha C V_{DD}^2 f \tag{4.2}$$

where α is defined as the switching factor (a metric measuring how often a node in the circuit switches), C is the total capacitance, V_{DD} is the voltage of the power supply, and f is the frequency of the clock.

4.2 Power Model Creation

A model for the power consumption of the oracle was created to train the surrogate model in order to analyze its effects on model extraction and adversarial attacks. Several assumptions were made to create the model. It is assumed that the target

network is implemented on hardware, and power consumption is able to be measured. In most settings, power consumption will likely be measured by an adversary while the oracle is in inference, so it is assumed that the parameters of the network will not change between subsequent inputs. To collect the power consumption, it is assumed that the attacker has access to the input. The oracle is also assumed to be running in ideal conditions, where the inputs passed into the network quickly and the delay between inputs is minimal, to reduce static power.

In this work, the primary target of the side channel attack is the activation state of the binary neurons in the hidden layer. It is assumed that the power consumption associated with the switching of the hidden neurons can be separated from the rest of the power components (e.g. synapses) by using knowledge of the propagation delays. The target activation function is the binary activation, where the output of the activation is 0 if the input is below 0.5, and 1 if it is above. The attacker collects the data for all nodes, or the activation map, in the activation layer. After the measurement is taken, the attacker introduces the next input, and measures the power consumption of the same activation layer. The attacker then is able to see the effects of the subsequent inputs in the activation map. As the network is on hardware, its power consumption adheres to trends observed in traditional CMOS circuits. As the neural network is constantly running, static power is low as there is little idling time. Since the network is running under ideal conditions, it can be assumed that most of the power consumption is switching power. As switching power consumption is only observed on transitions from 0 to 1, the attacker is able to estimate the total power consumption between the two inputs through a summation of the nodes that switched this way.

To simulate this in Keras, a copy of the trained oracle was taken and modified where the outputs of the copy was set to the activation layer. An input from the training set that will be used to train the surrogate is passed into the input layer,

and the activation map of the specific input is obtained. After the activation map of the next input is obtained, the number of transitions from low to high are counted to represent a unit of power consumption. This can be summarized by the following equation:

$$\mathbf{P(t)} = (\mathbf{x(t)} - \mathbf{x(t-1)}) \times \mathbf{x^T(t)} \tag{4.3}$$

where $\mathbf{x(t)}$ is the vectorized activation map of the current input, $\mathbf{x(t-1)}$ is the vectorized activation map of the previous input, $\mathbf{x^T(t)}$ is transpose of the current activation map, and $\mathbf{P(t)}$ is a vector containing all the nodes that transitioned from 0 to 1. The difference of the activation maps will produce values in the set [-1, 0, 1]. A value of -1 indicates a transition from 1 to 0 and a value of 0 implies no transition at all. A value of 1 indicates a transition of 0 to 1, implying that power was consumed. To filter out the -1 values, the transposed version of the activation map is multiplied to the difference.

Figure 4.1 shows a high-level diagram of the power consumption model.

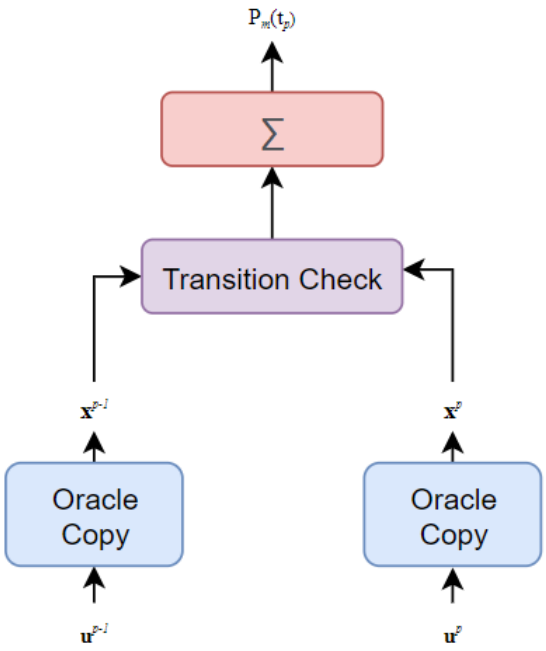


Figure 4.1: High-level diagram of power model.

$\mathbf{u^{p-1}}$ is the previous input and $\mathbf{u^p}$ is the current input. Both pass through the modified oracle copy to produce the activation maps $\mathbf{x^{p-1}}$ and $\mathbf{x^p}$. Using the activation maps, the number of transitions between from 0 to 1 are calculated, and summed to provide an approximation for the power. This power approximation is calculated between each subsequent input in the training set that will be used for the surrogate training. An example of a power trace, or measurements of the power consumption over time, can be seen in Figure 4.2.

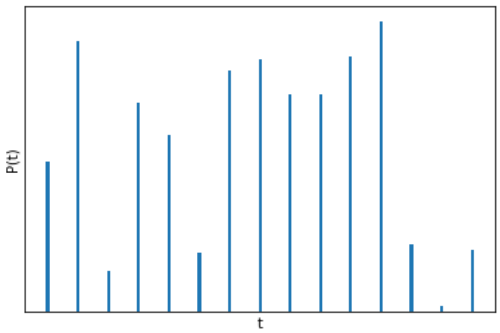


Figure 4.2: Example of power trace.

t is the time, and $P(t)$ is the power consumption that is measured during that time. Higher $P(t)$ values imply that more nodes in the circuit are switching, or transitioning from low to high.

Several other factors could be added to the power model to improve its accuracy. Adding in the other components of dynamic power, such as glitch and short-circuit power could be useful, but these may vary based on the hardware being used. No two pieces are hardware designed the same way, and thus will have a different layout, which can lead to different estimates of the power consumption related to the propagation delay caused by hazards. Short-circuit power can depend on the input transitions, which can be hard to predict, as there is no way to know how quickly an adversary passes inputs into the network. The background power consumption, as well as the power consumption of the synapses can be added to improve the overall model. By using an approximation of the switching power, the power model may be modified to adapt to other neural network architectures and perhaps even to hardware.

4.3 Application of Power Model

A Siamese-style network was designed to incorporate the side channel power data into the surrogate loss function [22], as shown in Figure 4.3.

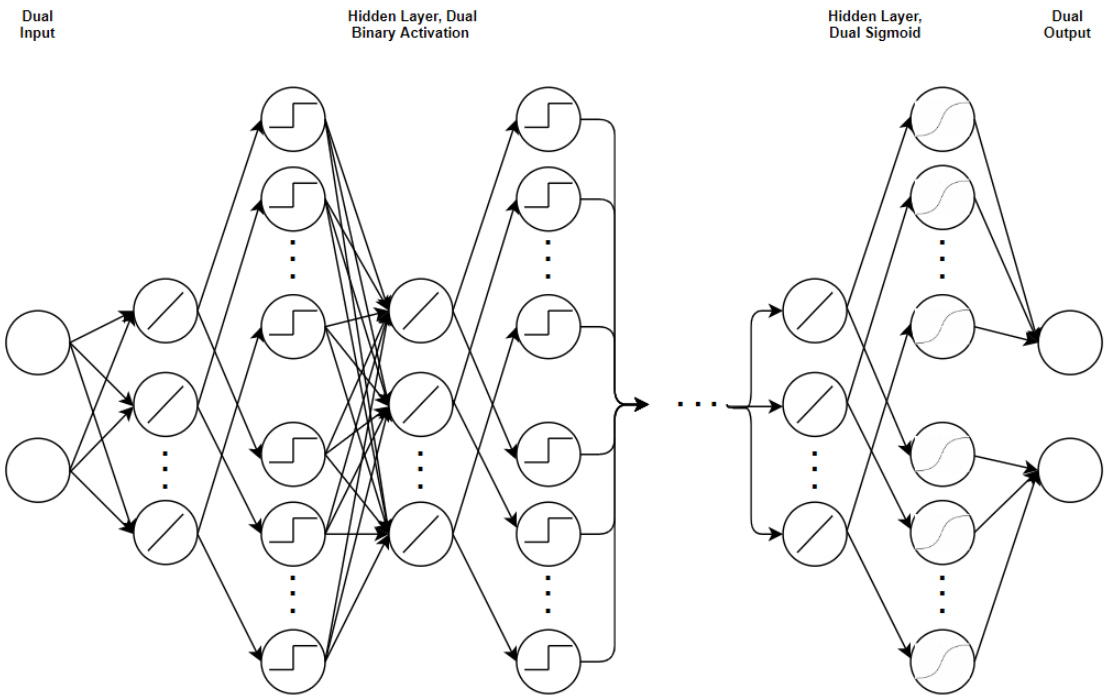


Figure 4.3: Surrogate model with dual input and output

Two separate inputs are connected to a fully connected layer that is the same size as the hidden layer in the oracle. The weight matrix between the input and hidden layer is the same for both inputs. The outputs for each individual input is then connected to its own activation layer. To obtain the output of the layer, the individual activation maps are both sent to another shared weight matrix in the output. The output of the last fully connected layer is sent to two separate softmax activations to produce the logits for the output. By having shared weight matrices, the dual-input model will have the same size weight matrices as the single-input model.

The power model detailed in Figure 4.1 was inserted into the dual-input model detailed in Figure 4.3. A high level diagram is shown in Figure 4.4.

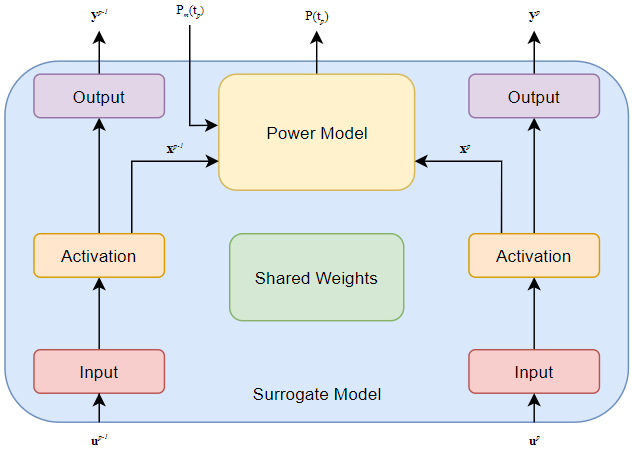


Figure 4.4: High-level diagram of network with power model.

$\mathbf{u^{p-1}}$ is the previous input and $\mathbf{u^p}$ is the current input. These pass through the dual-input surrogate model, which has a shared weight matrix in the hidden layer(s) that is the same size as the oracle. After the multiplication of the inputs and weights, the results are passed through two separate activation functions to obtain activation maps $\mathbf{x^{p-1}}$ and $\mathbf{x^p}$. These activation maps then go into two locations - the output layer, where they are processed as normal, and the power model developed earlier. The power consumption approximation from the oracle is also passed into the power model as a third output to train the surrogate.

In the power model, several neural network layers were used to mimic the counting of transitions that would cause power consumption. The activation maps are passed through a subtract layer to subtract one activation map from another. Depending on what the activation maps contained, the output of the subtract layer contained values of either -1 (meaning there was a transition from 1 to 0), 0 (meaning there was no transition), and 1 (meaning there was a transition from 0 to 1). To filter out only the 0 to 1 transition, a binary activation was used to ensure only the 1s passed through the activation. A small offset was used to mitigate any potential problems that may be caused at the boundary of the step function when the input was 1.

Finally, a custom layer was written to sum up the output of the activation, which is an approximation of the power consumption in the hidden layer of the surrogate model. Figure 4.5 shows the full dual-input model with the power model.

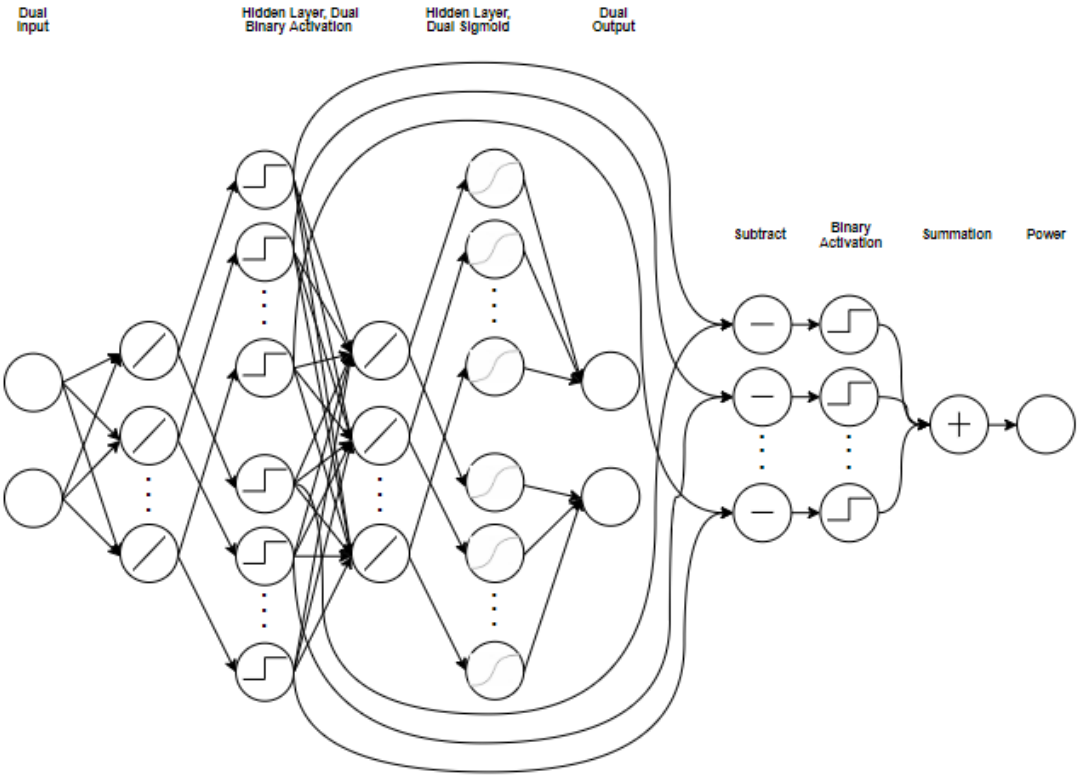


Figure 4.5: Power model with dual-input network.

The two subsequent inputs are passed into the dual input, which is then passed into the shared matrix for multiplication. After the hidden layer, the outputs of the multiplication are passed into the two activation layers - one for each input. After two activation maps are generated, they are passed into the output layer and the power model. Three separate outputs are generated - the two outputs corresponding to the two inputs, and the approximate power consumption of the surrogate model. To train the power consumption output of the surrogate, the estimated power consumption measured from the oracle was used. The power model itself does not have any weight matrices, so the number of parameters between the oracle and the surrogate with the power model will remain the same. To train the power model, a custom loss function

was used:

$$J_{PM} = \alpha J_{CE}(\theta, \mathbf{u^{p-1}}, \mathbf{y^{p-1}}) + \beta J_{CE}(\theta, \mathbf{u^{p}}, \mathbf{y^{p}}) + \gamma J_{MSE}(\theta, \mathbf{u^{p-1}}, \mathbf{u^{p}}, \mathbf{P(t)}) \quad (4.4)$$

where J_{CE} is the cross-entropy loss, J_{MSE} is the cross-entropy loss, $\mathbf{u^{p-1}}$ and $\mathbf{u^{p}}$ are the previous and current inputs, $\mathbf{y^{p-1}}$ and $\mathbf{y^{p}}$ are the previous and current outputs, and α, β, and γ are scaling factors. The MSE loss was used for the power consumption output data, and the cross-entropy loss was used for the outputs of the MNIST inputs.

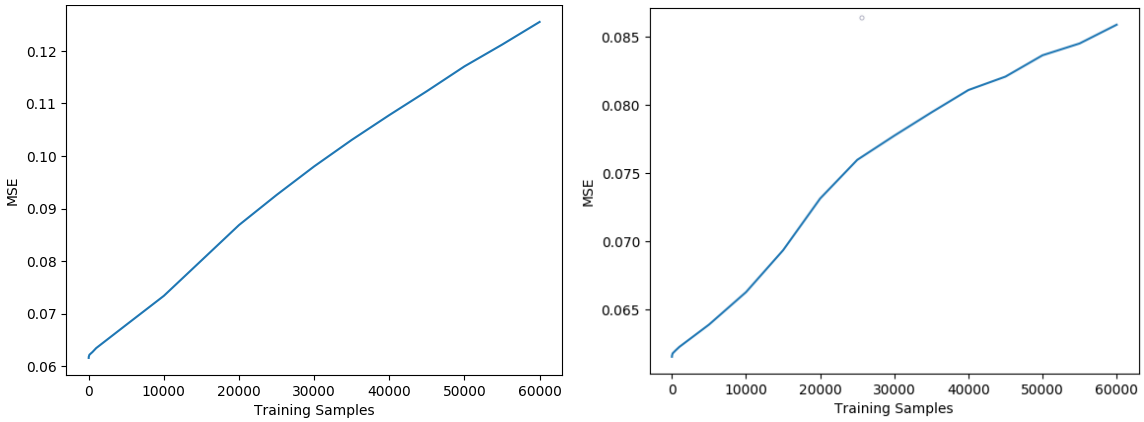
Chapter 5

Model Extraction and Adversarial Attacks With Power Information

Lastly, the simulations performed in Chapter 3 were re-run, but using the modified, Siamese-style network with the power information shown in Figure 4.5. The objective is to observe the effects the addition of the power information may have on the model extraction attack and the transferability of adversarial attacks from the surrogate model to the oracle. The setup between both sets of simulations remained the same, with the same perturbation factors and training set sizes for the surrogate model, with each training set size run 50 times and averaged. After the training of the oracle, its power consumption was estimated by capturing the number of nodes that switch between subsequent inputs. This switching power estimation was then used as an additional target output for the surrogate model. To generate the adversarial examples using the existing, single-input FGSM attack, the trained surrogate weights were transferred to another network of the same architecture as the oracle.

5.1 Model Extraction with Side-Channel Power Information

Figure 5.1 shows the MSE of the weights between the oracle and surrogate after the power information was added to the surrogate.

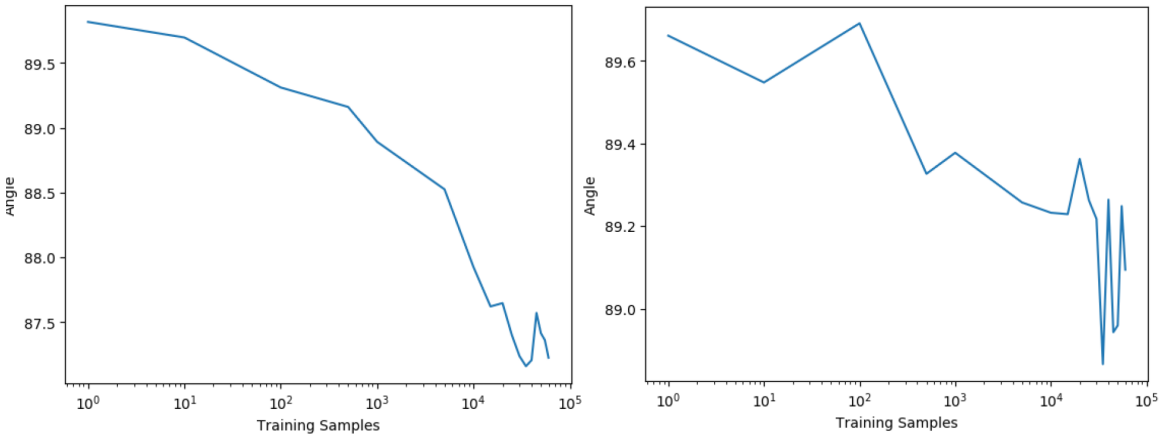


(a) Original **(b)** With Power Information

Figure 5.1: Mean-squared error of the weights between the oracle and the surrogate with and without power information.

The MSE grows linearly with surrogate training set size, similar to the case where no power information is used. However, larger training samples now contribute to lower MSE - while the previous model extraction had an average MSE as high as 0.13 for 60000 training samples, the MSE with the power information caps at an average of 0.087, for a improvement of up to 30%. The power information likely constrained the weight updates, as it was included in the loss function. At this point, it isn't clear whether the lower MSE is a result of the additional loss component of the power information, increased similarity of the oracle and surrogate models, or potentially both.

The angle between the weights was also analyzed, as shown in Figure 5.2.



(a) Original **(b)** With Power Information

Figure 5.2: Angle between the weights between the oracle and the surrogate.

The angle, however, did not change much after adding the power information. While the power loss may have constrained the weights to improve the MSE, an improvement of MSE does not necessarily constitute to improvement in angle measurement. For example, MSE calculations are affected by simply scaling one of the vectors, while the angle between two vectors will not be, as the calculations are done using unit vectors. Visually, while there could have been a decrease in the distance between the two vectors, the angle between the vectors does not necessarily have to decrease for this to occur.

5.2 Adversarial Attacks With power information

Figures 5.3 and 5.4 show the adversarial images and noise generated with the addition of the power information.

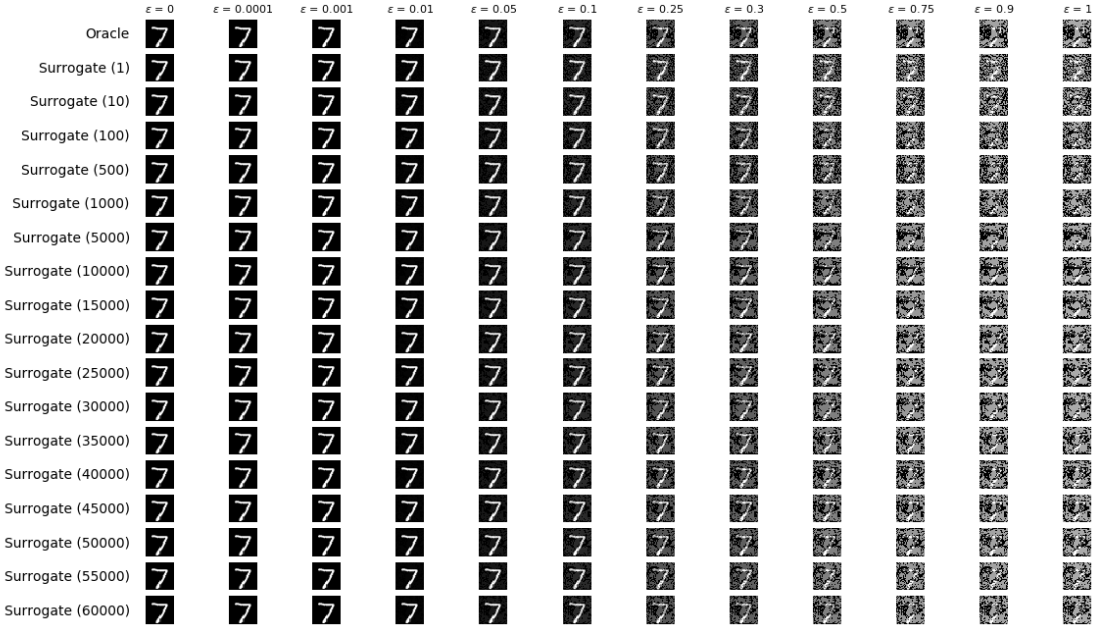


Figure 5.3: Sample adversarial image generated by FGSM with power.

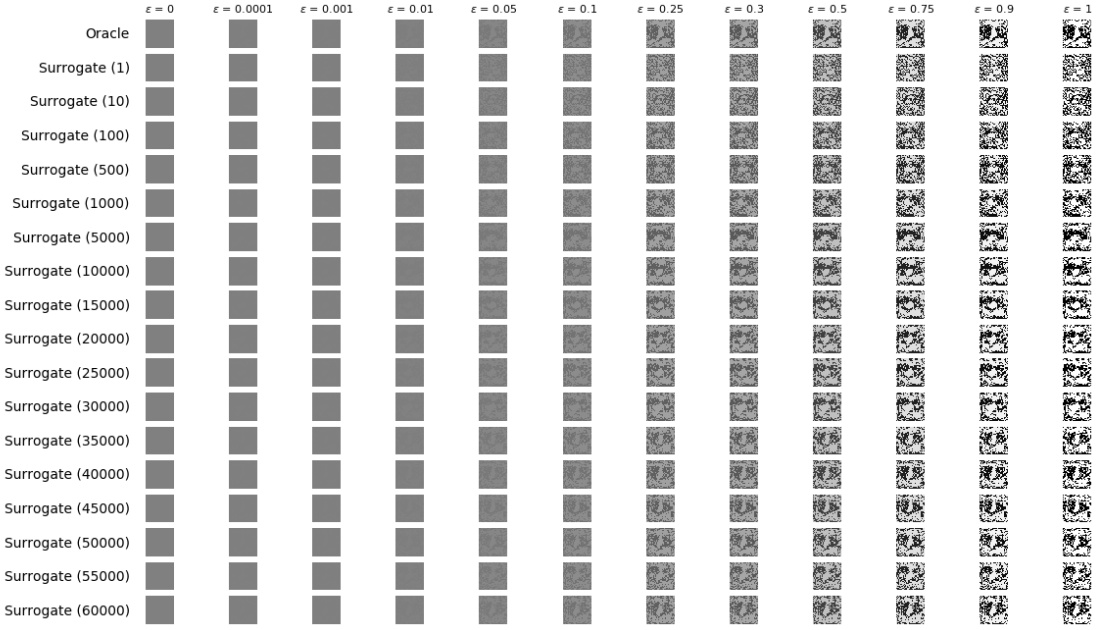
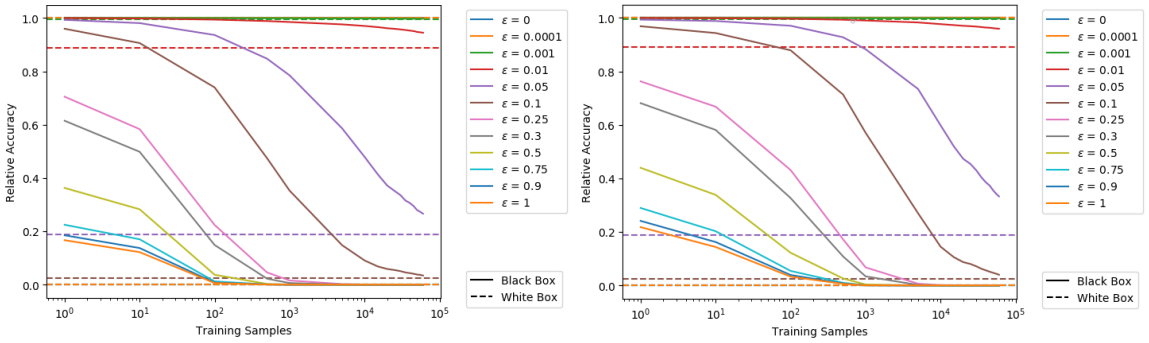


Figure 5.4: Sample noise added to images by FGSM with power.

Visually, the overall trend remained the same between the models with and without power - higher training examples produce adversarial images that are closer to what the oracle produced. Visually, however, the noise that was added from the

power information surrogate is different from the noise added without the power information. This implies that the signs of the gradient with respect to the input image were different.

Figure 5.5 shows the relative accuracies of the whitebox attack on the oracle and the blackbox attack of images generated from a power trained surrogate on the oracle.

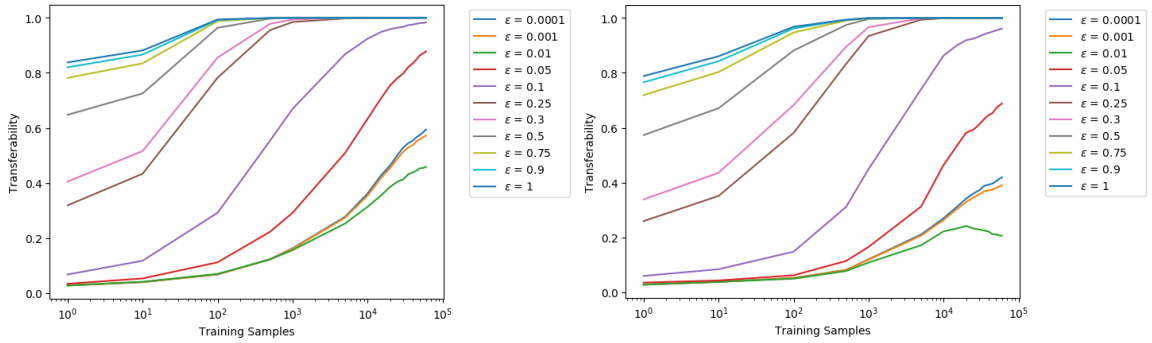


(a) Original **(b)** With Power Information

Figure 5.5: Whitebox vs. blackbox relative accuracies with power information.

Similar to the simulations without power, lower ϵ values so little to no changes due to the lack of noise added to the image, with the first noticeable change being at $\epsilon = 0.01$. The curves follow the same behavior where the blackbox curves are higher than or tangent to the whitebox line. However, for the perturbation factors in the middle of the spectrum, the blackbox attacks have a higher relative accuracy than the models without power, such as at 0.01 and 0.05, at higher training examples. This implies that one of the two networks is slightly more resistant to adversarial images in more complex models.

Figure 5.6 shows the transferability of attacks from the surrogate to the oracle.



(a) Original **(b)** With Power Information

Figure 5.6: Transferability of the FGSM attack with and without power information.

The transferability of attacks slightly increased at lower training set sizes, but noticeably decreased at higher training examples. This implies that either the surrogate or the oracle are more resistant against attacks, as transferability measures attacks that affect both. The attack breakdown gives a clearer picture on which network is more resistant overall. One overall improvement is that the lower perturbation strengths has transferabilities that increased linearly at lower training set sizes, whereas the transferability of the model without the power information increased at a slower rate.

Figure 5.7 and 5.8 show the attack breakdowns for various ϵ values with the power information.

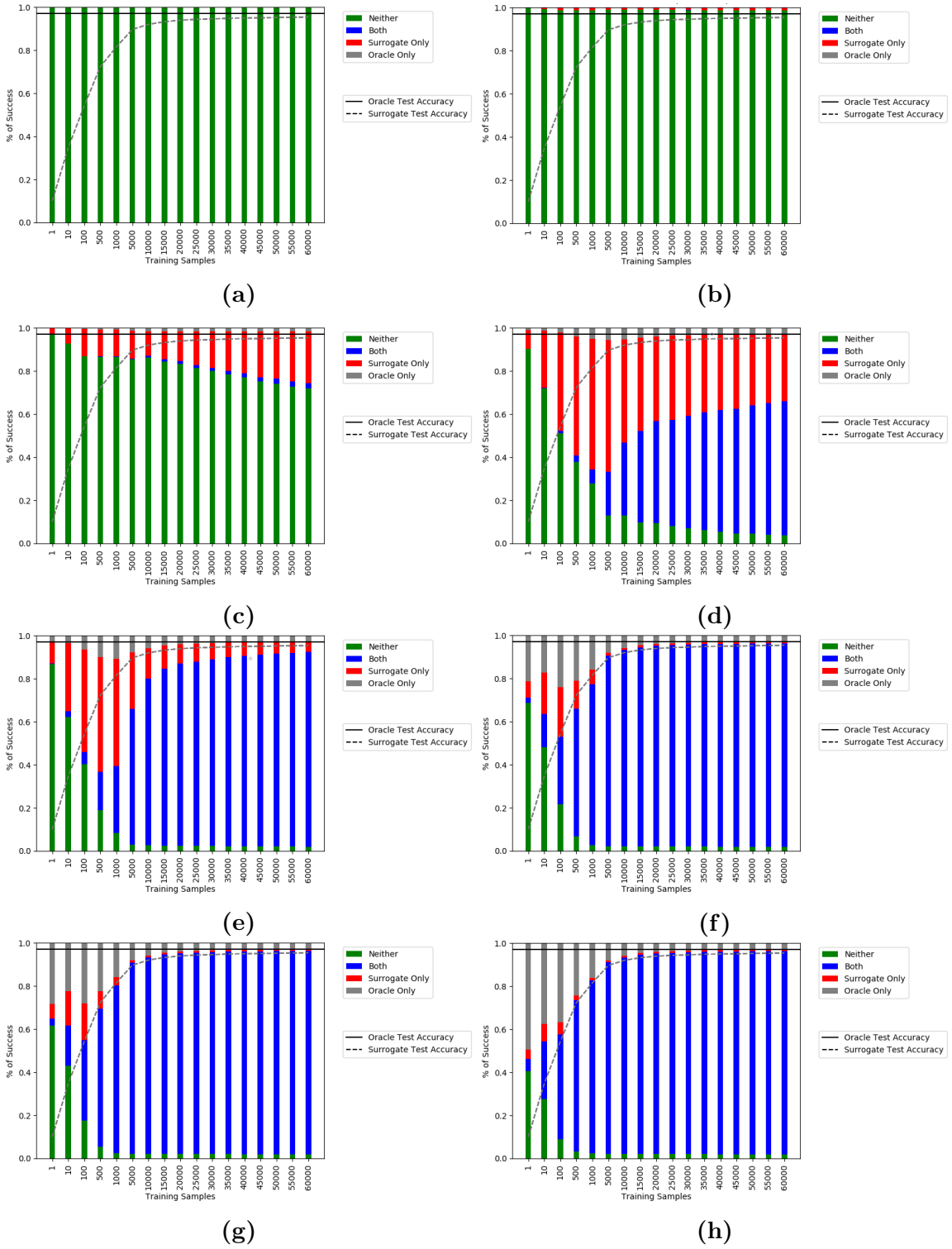


Figure 5.7: Attack breakdown for various ϵ values with power.

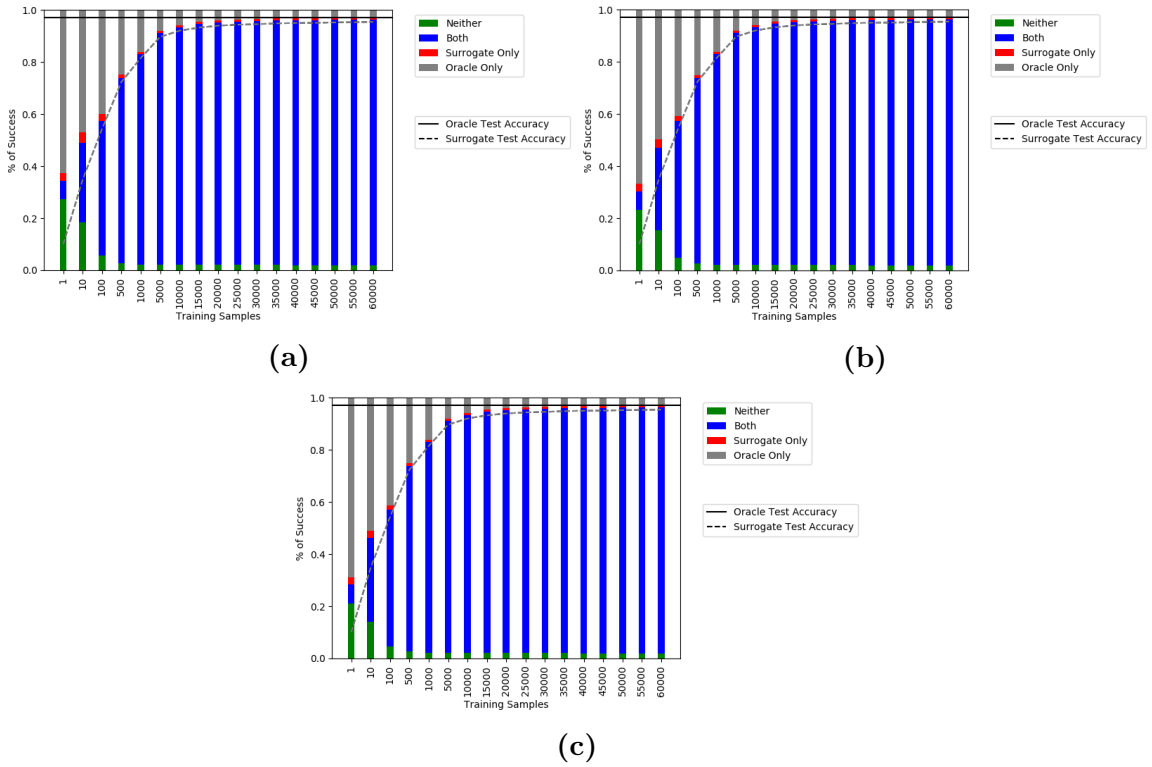


Figure 5.8: Attack breakdown for various ϵ values with power.

The overall breakdowns remained relatively similar to the ones of the attack on the model without power information, with one slight difference - the percentage of attacks successful on the oracle was slightly higher, and is more noticeable at higher epsilon values. This implies that the surrogate is in fact more resistant to attacks - the transferability overall suffered as the attack was never transferred to begin with. One reason for this could be the result of gradient masking from the input, as suspected by the authors of [23]. The main form of gradient masking comes from the non-smooth nature of the gradients of the binary activation function. This causes a regularization effect that causes the MLP to learn the decision boundaries of the training set better, which makes it overall more resistant to adversarial attacks [23].

Chapter 6

Conclusion & Future Work

In this work, a power model was developed to attempt to improve model extraction and adversarial attacks on binarized neural networks. While the power model did reduce the distance between the oracle and surrogate model parameter vectors in the weight space, the behavior against an adversarial example between the oracle and surrogate models were still dissimilar, despite the high test accuracies present from both models. Functional equivalency does not necessarily lead to equivalent model parameters, as shown in the lack of transferability of the adversarial attacks when the power model was added. One potential reason for the parameters not being equivalent is the problem of an under-determined system. As most state of the art model extraction attacks have shown, the functionality of a network can be reproduced, as there exists many possible solutions for a specific task. However, to reproduce the parameters, there exists a small set of solutions - the exact parameters of the oracle model and any permutations of those parameters. It is likely that while attempting to find the minimum while training, the minima corresponding to the exact oracle weights was extremely narrow, and therefore, a local minima was settled into instead of the minimum.

A new metric to relate how "close" two networks are could also be explored. For the most common model extraction attacks, such a metric would not be possible as the network topologies are not the same. For a high-fidelity extraction, however,

metrics such as MSE and angle may not be enough to say how similar the networks are. The metric would need to account for the potential millions of parameters in deeper neural networks.

Several options could be considered to potentially improve the power model. Introducing more components, such as power that could be consumed by the multiply and accumulate operation, into the power model could make it more robust and accurate. Using real hardware instead of an estimated model can make the attacks much more effective, considering an estimate alone helped reduce the MSE for the full training set by 30%. Since the inputs are known, establishing a correlation between what the inputs and the output of the activation could provide a more detailed power trace.

The model could be used on more complex networks, such as MLPs with two or more hidden layers, convolutional neural networks, and perhaps even recurrent neural networks. For multi-hidden layer MLPs, a way to factor in all activation maps at once will need to be explored, as the current version of the model only uses the power consumption measured from a single hidden layer. The main challenge of modelling the power of a convolutional neural networks is that the overall power consumption of the convolution layer(s) would be quite large compared to the other layers of the network, as the convolution operation is computationally intensive. The power consumption of the activation function and the max pooling layers would be overshadowed, which would require a large modification of the power model at its current state.

There is also the question of defending these attacks. For a hardware based neural network, this may be a bit difficult to achieve, as obscuring the details of the network will not prevent potential adversaries from utilizing side channel information to their advantage. One potential defense, specifically for the style of attack outlined in this work, would to incorporate a CMOS circuit that would have transitions between 0 and 1 no matter what the inputs are, such as domino logic [21]. For FGSM attacks

in general, adversarial training can work, but may not be as effective for stronger adversarial attacks. Utilizing dropout can also make the oracle topology harder to determine, as nodes are dropped during the training process, which can complicate power measurements. The introduction of Bayesian logic to the network could also make it more robust by introducing a probabilistic model while training the neural network. [24].

Bibliography

[1] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2015.

[2] J. Troung, P. Maini, R. Walls, and N. Papernot, "Data-free model extraction," November 2020.

[3] L. Wei, Y. Liu, B. Luo, and Q. Xu, "I know what you see: Power side-channel attack on convolutional neural network accelerators," March 2018.

[4] L. Batina, S. Bhasin, D. Jap, and S. Picek, "Csi neural network: Using side-channels to recover your artificial neural network information," October 2018.

[5] N. Papernot, P. McDaniel, I. Goodfellow, J. Somesh, Z. Berkay Celik, and A. Swami, "Practical black-box attacks against machine learning," March 2017.

[6] M. Jagielski, N. Carlini, D. Bethelot, A. Kurakin, and N. Papernot, "High accuracy and high fidelity extraction of neural networks," March 2020.

[7] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training neural networks with weights and activations constrained to +1 or -1," March 2016.

[8] B. Zhuang, C. Shen, M. Tan, and I. Reid, "Structured binary neural networks for accurate image classification and semantic segmentation," November 2018.

[9] A. Shafahi, W. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, "Poison frogs! targeted clean-label poisoning attacks on neural networks," November 2018.

[10] A. Demontis, M. Melis, M. Pintor, M. Jagielski, B. Baggio, A. Oprea, X. Rotou, and F. Roli, "Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks," June 2019.

[11] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Srndic, P. Laskov, G. Giorgio, and F. Roli, "Evasion attacks against machine learning at test time," August 2017.

[12] J. Xu, Z. Cai, and W. Shen, "Using fgsm targeted attack to improve the transferability of adversarial example," December 2019.

[13] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," February 2017.

[14] N. Carlini, M. Jagielski, and I. Mironov, "Cryptoanalytic extraction of neural network models," March 2020.

[15] T. Takemura, N. Yanai, and T. Fujiwara, "Model extraction attacks against recurrent neural networks," February 2020.

[16] Y. Zhou and D. Feng, "Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing," 2005.

[17] R. Zhao, W. Song, W. Zhang, T. Xing, J. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating binarized convolutional neural networks with software-programmable fpgas," February 2017.

[18] K. Yoshida, T. Kubota, M. Shiozaki, and T. Fujino, "Model-extraction attack against fpga-dnn accelerator utilizing correlation electromagnetic analysis," 2019.

[19] X. Hu, L. Liang, L. Deng, S. Li, X. Xie, and Y. Ji, "Neural network model extraction attacks in edge devices by hearing architectural hints," March 2019.

[20] W. Hua, Z. Zhang, and G. Suh, "Reverse engineering convolutional neural networks through side-channel information leaks," November 2018.

[21] N. Weste and D. Harris, *CMOS VLSI Design: Circuits and Systems as Perspective*. Addison-Wesley, 2010.

[22] S. Berlemont, G. Lefebvre, S. Duffner, and C. Garcia, "Class-balanced siamese neural networks," 2018.

[23] A. Galloway, G. Taylor, and M. Moussa, "Attacking binarized neural networks," January 2018.

[24] S. Arangio and J. Beck, "Bayesian neural networks for bridge integrity assessment," November 2010.